美濃研夏のセミナー2002

ver 1.1, 2002/08/26

まだ非公開版・原稿訂正受付中

日程

2002/08/24-27

場所

名前 :京大ヒュッテ
場所 :長野県志賀高原
住所 :長野県下高井郡山内町志賀高原蓮池
電話 :0269-34-2105

テキスト

Title: `Ontology Learning for the Semantic Web'
Author: Alexander Maedche
Code: ISBN 0-7923-7656-0
1章から8章まで(P.1 - P.199)

参加者

美濃(職)、八木(職)、亀田(職)、 西口(職)、
飯山(D3)、伊藤(D2)、山肩(D1)、 身野(M2)、新津(B4)、
仙田(OB)、神嶌(OB)、上田(OB)、先山(OB)、北脇(OB)、村上(OB)、須藤(OB)、古村(OB)

幹事

伊藤

レジュメ

Chapter 1-2:八木
Chapter 3:身野
Chapter 4:亀田
Chapter 5:伊藤
Chapter 6:山肩
Chapter 7:西口
Chapter 8:飯山
Stumme et al., 2000 (P.91) の参考文献を末尾に添付

参考

http://www.kameda-lab.org/research/imss/index-j.html

第1章　序論 (担当：八木)

1.1　動機と問題記述

ウェブ上では，いまのところシンタクスをもとに検索することしかできない．コンテンツの意味を理解したうえで情報を引き出したりウェブを維持管理したりするのはすべて人手によっている．そこで提案されたのが「セマンティックウェブ」(Tim Berners-Lee)

セマンティックウェブ コンテンツの意味を理解するために必要な背景知識を，機械処理可能なメタデータとして蓄積しておく．これを利用してさまざまな自動サービスを実現するものがセマンティックウェブ．またそこで利用される背景知識を体系的にまとめたものがオントロジー．

オントロジー ある分野に関する包括的な概念をフォーマルに体系化したもの．データの意味を明示的に記述するメタデータ．セマンティックウェブの基礎となる．

オントロジーの構築 人手で組み立てるのは煩雑なうえ，うまく組み立てられなくて知識獲得のボトルネックになることが多い．工学的センスでいうと，セマンティックウェブ実現のためには，さまざまな分野のオントロジーを容易に構築できる必要がある．そこで，知識工学で研究・実証されてきた機械学習の技術を応用する．オントロジー工学．

オントロジー工学におけるバランス協調モデリング オントロジーを構築するため，さまざまなデータに対して機械学習技術を適用する．ただし，要所要所で人間が介入し，半自動的に構築を行う．完全に自動化するのはまだまだ先のこと．

1.2　研究課題

本編の前振りとして，イモヅル式に課題を羅列する．知識獲得に機械学習を使う (use) のは目新しいことではないが，知識獲得と機械学習を結合 (combine) させるのは難しい（なんのこっちゃ）．

- 人間は機械学習のプロセスにどのように介入すればよいのか？機械学習によって，どれだけオントロジー工学の複雑さを低減できるのか？
- どのようにしてオントロジーに含まれる定義を自然言語と結びつけるのか？どのようにしてオントロジーをセマンティックウェブに実装するのか？
- オントロジーを学習するのに必要なコアの要素とは何か？オントロジー学習にはどんなフェーズが存在するのか？(要するにオントロジー学習にはきちんとした枠組みがない)
- テキストデータのように構造化されていないデータから HTML や辞書のように半構造化されたデータまで，どのデータからどんな知識が得られるのか？いまあるウェブのデータからオン

トロジーを抽出して維持管理していくためには，どのデータが必要でどのような処理をしなくてはならないのか？

- オントロジー学習がうまく働くようにするためには，情報をどのようにテキストに記述すればよいのか？オントロジー学習をサポートするためにはどのような技術・アルゴリズムを適用すればよいのか？
- これまで機械学習で提案されてきた知識獲得アルゴリズムをどのようにオントロジー学習に適用すればよいのか？機械学習による適応がオントロジー学習のどこに求められているのか？
- オントロジー学習の結果はどのようにすれば評価できるのか？どのようにすれば二つあるオントロジーをフォーマルにきちんと評価できるのか？
- オントロジー工学において，機械学習技術と比べて人間のパフォーマンスはどの程度のものか.

1.3　ガイド

この本は 4 部構成になっている.

1. 基礎知識 (1,2,3 章)
2. セマンティックウェブのためのオントロジー学習 (4,5,6 章)
3. 実装と評価 (7,8 章)
4. 関連研究と展望 (9,10 章)

（各章についての説明はそれぞれの担当者にお任せするということで，割愛）

図 1.1： この本の読み方.

2 章では，オントロジーと知識ベースの構造を形式的に定義する．3 章ではオントロジー工学全体の枠組みについて述べてある．オントロジー工学をすでに熟知している人は 3 章をとばして 4 章へ．4 章ではオントロジー学習全体の枠組みについて述べてある．オントロジー学習のツールについてだけ興味がある読者は，データ処理やアルゴリズムなど技術的なことが述べられている 5,6 章はとばして 7 章へ．7 章はオントロジー学習ツールの実装と評価について書いてあり，8 章はオントロジーの比較と評価について書かれてある．関連研究と展望に興味がある読者は 9 章を読めばよい．10 章では結論を述べるとともに，積み残した問題をリストにしてある.

第2章　オントロジー　定義と概要

この章のトピックは，1) オントロジーの起源，2) オントロジー構造の形式的定義，3) 知識ベース構造の形式的定義，4) オントロジーの分類，5) オントロジーを利用したアプリケーション．

存在論 (Ontology) 事物の性質と構成に関する哲学の一分野.「存在するとはどういう事か」「存在するものすべてに共通する性質とは何か」

オントロジー (ontology) 世界についてのある見方を説明する概念体系．計算機科学では，事実を記述するための語彙とその意味に関する仮定集合からなる．一般に仮定は一階の述語論理式で表され，語彙は概念と呼ばれる単項述語と関係と呼ばれる二項述語として表される．

計算機科学におけるオントロジーへの要請　人間は自然言語を使って世界を記述しているが，曖昧なので計算機科学でモデルを立てるには不適切．そこで形式言語を考える．数学を形式言語として見たときの基本的な性質について研究したのがフレーゲ．ポイントは言語と命題の分離にある．

形式言語は，チョムスキー流の項書き換えシステムとして定式化される．無限の表現を生み出す有限の文字セットと有限の生成規則で言語が定義されている．しかし，文法的に正しい言語を生成できても，それは生成される文章の意味を捕まえたことにはならない．

文法的に正しい表現とそれが意味する概念との間にある「意味のギャップ」を橋渡しするのがオントロジー．一般に同じ情報を記述するのでもシステムが違えば使う言語も異なるため，複数のシステムを跨いで知識を共有したり再利用するのは難しいが，この問題もオントロジーで解決できる．

オントロジー　その起源　プラトンとアリストテレス．

プラトン モノに正しく名前を付けるというのはどういうことか．あらゆるモノに正しく名前が付けられた「最適な世界」では，ある言語表現はすべての人にとって同じモノを意味する．しかし，ものごとは常に変化しているのでそれに完璧な名前を付けるのは無理なのではないか．

アリストテレス 単なる単語の定義にとどまらず，実体の「エッセンス」を記述することでどのようなモノであるかを明確に説明しようとした．種と種差を決定することで定義を与え，その定義のうえで論理変数を使って個別の事例を分類する．類似した個体に共通する性質を明らかにし，その性質を定義に照らすことでそれらがどうして同じ分類に属するかがわかる．

アリストテレスの論拠は不十分．言語による意味の伝達には,「感覚」の違いによる意味の違いから生じる曖昧さなど，避けられない限界があることを無視している．フレーゲはそれをふまえて「意味の三角形」を提案した．

図 2.1 意味の三角形.

記号 (symbol) が指示対象 (referent) をいきなり表象しているわけではない．記号から指示対象への矢印が破線になっているところがポイント．記号が概念 (concept) を喚起し，その概念が注意を指示対象に向けさせた結果，人間はその記号がある指示対象を表象していると考える．

現実世界では，複数の記号がひとつの指示対象を表象していることがある．またひとつの記号が複数の指示対象を表象しているために曖昧性が生じることもある．

フレーゲによる古典的な例は「明けの明星」「宵の明星」と「金星」の話．どちらも物理的な指示対象は金星のことだが，昔は同じものだと思われていなかった．これについては，物理的な指示対象が同じだから同じだということもできるし，それぞれの記号が意味するところは単に物理的な指示対象だけでなくそれにまつわるいろんな情報も含まれているので異なるということもできる．例えば「明けの明星」というからにはそこに上ってくる太陽の存在は欠かせない．また夜明けをみたことがない人がいるとしてその人が「明けの明星」と聞けば，そこに金星とは別の星が存在することがわかったりする．つまりどれも金星に関係はしているが，そこから喚起される概念は個人の背景知識に依存するということ．

2.1　コミュニケーションのためのオントロジー　階層化アプローチ

オントロジーでは，世界に存在する対象を記述する符号がフォーマルに定義され，符号から対象への可能な限り正確なマッピングを実現する．知識ベースは，特定の環境を記述するものとしてオントロジーの上位層に定義される．

図 2.2 オントロジーを使ったコミュニケーション．

この図は人間あるいは機械エージェントのコミュニケーションの全体像で，3 層構造になっている．1) まず実世界に存在するモノを考える．ここでは，人間・機械エージェント・車・ジャガーの 4 つ．2) 次に，人間同士あるいは機械エージェント同士がやりとりする記号とその文法を考える．これが 3 層の最上位層となる．3) 最後に，記号に関連づけられる概念と意味構造をもったモデルの分析が行われる．

まず，与えられたオントロジーを使わないことを考える．人間の間で「ジャガー」という符号をやりとりするとして，それぞれが勝手に自分の中で概念を喚起する．それらの概念は車のジャガーに関連づけられることもあるだろうし，動物のジャガーかもしれない．同じ事は機械エージェントについてもいえる．

これに対して，人間が両方とも同じオントロジーを使うとしたら，同じモノを指示対象とする確率は高くなる．機械についても同じこと．違うオントロジーを使って間違った指示対象を選んでしまったら，指示対象を絞り込むために推論すればよい．例えば，動物には車輪がないが車にはあるので，そこを確認してみればよい．

「ジャガー」といって，車と動物の和集合を指すものとすることもできる．もしくは,「動物ージャガー」と「車ージャガー」というふたつの概念を指すものということもできる．どちらかというと後ろの方がおすすめ．いずれにしてもヘンな話だが,「動物ージャガー」といった符号を見せる方が,(「動物ージャガー」と「車ージャガー」の和）なんて概念を持ち出すよりもマシ．

オントロジーベースのシステムのユーザも開発者もオントロジー構造から利益を受けることができる．論理的な厳密さに加えて，考えられる誤解を説明できるし自然言語へ直接マッピングすることもできる．

オントロジー構造 $\mathcal{O}$　ここで定義するオントロジー構造は，OKBC モデルとか ISO 規格など，用語の原理や方法論がすでに分析されているものの拡張．人間は互いに記号をやりとりしながらその定義を拡張してきたわけで，意味はコミュニケーションの中で作り出されてきたといえる．オントロジー学習のアプローチでは，そういったコミュニケーションで人間が交換してきた記号を使っていることから,「意味の発見」は黙示的にデータの中に含まれていると考える．従って，自然言語と形式意味との関係が重要になる．

求められるオントロジー構造について述べるにあたって記号論を起点にする．ピアースによると，記号論では以下の 3 つが区別される．

統語論：文法 (syntax) 記号間の関係についての研究.

意味論：意味 (semantics)　記号とそれが記述する実世界の事物との関係を分析する.

語用論：用法 (pragmatics)　ある目的に対して記号がどのように使われるかを分析する.

3 つのレベルは独立ではない．例えば文法と意味の間は，記号と言明との参照関係によってつながっている．ここではオントロジーを，機械と人間の間で意味的な対話をするのに使われるモデルだとする．記号論的にみたオントロジー構造は 2.1 のように定義できる.

定義 2.1　オントロジー構造 $\mathcal{O}$ は, 5 つ組 $\mathcal{O} := \{\mathcal{C}, \mathcal{R}, \mathcal{H}^{\mathcal{C}}, rel, \mathcal{A}^{\mathcal{O}}\}$ で表される.

- 排他集合 $\mathcal{C}$ と $\mathcal{R}$ の要素をそれぞれ概念 (concept) と関係 (relation) と呼ぶ.
- 概念階層 $\mathcal{H}^{\mathcal{C}}$:$\mathcal{H}^{\mathcal{C}}$ は，$\mathcal{H}^{\mathcal{C}} \subseteq \mathcal{C} \times \mathcal{C}$ なる有向関係で，概念階層ないし分類と呼ばれる．$\mathcal{H}^{\mathcal{C}}(C_1, C_2)$ は，C_1 が C_2 の下位概念であることを示している.
- 関数 rel ：概念間を非分類的 (non-taxonomic) に関連づける $\mathcal{R} \to \mathcal{C} \times \mathcal{C}$ なる関数．また，$rel(R) = (C_1, C_2)$ は $R(C_1, C_2)$ とも書く.
  - 関数 dom: R のドメインを与える $\mathcal{R} \to \mathcal{C}$ なる関数で，$dom(R) := \Pi_1(rel(R))$ で定義される.
  - 関数 $range$: R のレンジを与える $\mathcal{R} \to \mathcal{C}$ なる関数で，$range(R) := \Pi_2(rel(R))$ で定義される.
- オントロジーの公理集合 $\mathcal{A}^{\mathcal{O}}$．これは一階述語論理など適切な論理言語で表される.

このオントロジー構造をこの本では使っていく．これは簡明で納得のいくものだし，既存のオントロジー記述言語にも容易にマッピングできるだろう.（ほんまかいな）ただし上の定義には，自然言語とオントロジーをつなぐための語彙レベルについての明示的な表現が欠けている．そこで以下のように語彙を定義する.

定義 2.2　オントロジー構造 $\mathcal{O} := \{\mathcal{C}, \mathcal{R}, \mathcal{H}^{\mathcal{C}}, rel, \mathcal{A}^{\mathcal{O}}\}$ の語彙 $\mathcal{L}$ は 4 つ組 $\mathcal{L} := \{\mathcal{L}^{\mathcal{C}}, \mathcal{L}^{\mathcal{R}}, \mathcal{F}, \mathcal{G}\}$ で表される.

- 集合 $\mathcal{L}^{\mathcal{C}}$ と $\mathcal{L}^{\mathcal{R}}$ を，それぞれ概念と関係の語彙項目 (lexical entry) と呼ぶ.

- $\mathcal{F} \subseteq \mathcal{L}^{\mathcal{C}} \times \mathcal{C}$ なる関係 $\mathcal{F}$ と，$\mathcal{G} \subseteq \mathcal{L}^{\mathcal{R}} \times \mathcal{R}$ なる関係 $\mathcal{G}$ を，それぞれ概念と関係への参照 (reference) と呼ぶ．$\mathcal{F}$ について，$L \in \mathcal{L}^{\mathcal{C}}$ とすると，

$$\mathcal{F}(L) = \{C \in \mathcal{C} | (L, C) \in \mathcal{F}\}$$

$$\mathcal{F}^{-1}(C) = \{L \in \mathcal{L}^{\mathcal{C}} | (L, C) \in \mathcal{F}\}$$

  同様にして $\mathcal{G}$ と $\mathcal{G}^{-1}$ も定義される．

一般に，ひとつの語彙項目は複数の概念もしくは関係を参照し，ひとつの概念ないし関係は複数の語彙項目から参照される．

オントロジー構造の例　概念集合 $\mathcal{C} := \{x_1, x_2, x_3\}$，関係集合 $\mathcal{R} := \{x_4\}$ がある．そこに概念階層 $\mathcal{H}^{\mathcal{C}}(x_2, x_1)$ があって，非分類的関係 $x_4(x_2, x_3)$ が定義されているものとする．また語彙は以下のように定義されているとする．

$$\mathcal{L}^{\mathcal{C}} = \{\text{"Person"}, \text{"Employee"}, \text{"Organization"}\}$$
$$\mathcal{L}^{\mathcal{R}} = \{\text{"works at organization"}\}$$

さらに以下のような参照が定義されているとする．

$$\mathcal{F}(\text{"Person"}) = x_1$$
$$\mathcal{F}(\text{"Employee"}) = x_2$$
$$\mathcal{F}(\text{"Organization"}) = x_3$$
$$\mathcal{G}(\text{"works at organization"}) = x_4$$

このオントロジー構造を図に描くと，図 2.3 のようになる．

ここまでででは，プリミティブの意味については詳しく述べてこなかったが，次章ではこのオントロジー構造が別のかっちりした記述言語（F-Logic とか W3C の RDF とか）でどのように実現されるかを詳述する．そこでは Staab らが提案してきたセマンティックパターンアプローチが使われることになるだろう．

セマンティックパターンはセマンティックウェブの開発者間のコミュニケーションにも使われるだろうし，異なる記述言語への移植にも使われる．さらに異なる表現間の橋渡しになるし，異なった知識モデリング手法の橋渡しにもなる．セマンティックパターンはスクラッチから起こすのではなく，ソフトウエア工学や知識表現研究に関する知見から抽出されて利用される．セマンティックパターンの詳細は次章にて．

知識ベース構造 $\mathcal{KB}$　オントロジーと知識ベースの以下のような違いをもとに，知識ベース構造 $\mathcal{KB}$ を定義する．

オントロジー	知識ベース
対象領域の概念構造をつかまえたい．	既知のある状態を特定したい．
内包的 (intensional) な論理定義．	外延的 (extensional) な部分からなる．
システム構築時に構築されて変化しない．	蓄積される事実は常に変化する．

定義 2.3 知識ベース構造 $\mathcal{KB}$ は，4 つ組み $\mathcal{KB} := \{\mathcal{O}, \mathcal{I}, inst, instr\}$ によって定義される．

- オントロジー $\mathcal{O} := \{\mathcal{C}, \mathcal{R}, \mathcal{H}^{\mathcal{C}}, rel, \mathcal{A}^{\mathcal{O}}\}$
- 実体 (instance) 集合 $\mathcal{I}$
- 関数 $inst$: $\mathcal{C} \rightarrow 2^{\mathcal{I}}$ なる概念実体化関数．また $inst(C) = I$ を $C(I)$ とも書く．
- 関数 $instr$: $\mathcal{R} \rightarrow 2^{\mathcal{I}\times\mathcal{I}}$ なる関係実体化関数．また $instr(R) = \{I_1, I_2\}$ を $R(I_1, I_2)$ とも書く．

さらに以下のように語彙を定義する．

定義 2.4 知識ベース構造 $\mathcal{KB} := \{\mathcal{O}, \mathcal{I}, inst, instr\}$ の語彙 $\mathcal{L}^{\mathcal{KB}}$ は 2 つ組 $\mathcal{L}^{\mathcal{KB}} := (\mathcal{L}^{\mathcal{I}}, \mathcal{J})$ で表される．

- 集合 $\mathcal{L}^{\mathcal{I}}$ は，実体集合 $\mathcal{I}$ の語彙項目 (lexical entry)．
- $\mathcal{J} \subseteq \mathcal{L}^{\mathcal{I}} \times \mathcal{I}$ なる参照 (reference)．$\mathcal{J}$ について，$L \in \mathcal{L}^{\mathcal{C}}$ とすると，

$$\mathcal{J}(L) = \{I \in \mathcal{I} | (L, I) \in \mathcal{J}\}$$

$$\mathcal{J}^{-1}(I) = \{L \in \mathcal{L}^{\mathcal{I}} | (L, I) \in \mathcal{J}\}$$

一般に，ひとつの語彙項目は複数の実体を参照し，ひとつの実体は複数の語彙項目から参照される．

特定のセマンティックウェブ向け言語による知識ベースの実現については，次章にて．オントロジーと知識ベースは別々に定義されているが，これらは特に区別がない．対象となるドメインやモデル屋の見方やモデル屋の経験によって，実体や関係がオントロジーや知識ベースに入れられたり入れられなかったりというようなことはまれである．オントロジーと知識ベースの関係についての全体像は次章の図 3.1 にて．

オントロジー構造の例 先の事例で示したオントロジー構造があるものとして，実体集合を $\mathcal{I} := \{i_1, i_2\}$ とする．概念実体化関数を $x_2(i_1)$,$x_3(i_2)$ とし，関係実体化関数を $x_4(i_1, i_2)$ とする．語彙項目は $\mathcal{L}^{\mathcal{KB}} := \{$"person:ama","organization:unika"$\}$ と定義され，語彙項目から実体へのマッピングは $\mathcal{J}$("person:ama") $= i_1$ と $\mathcal{J}$("organization:unika") $= i_2$ として定義される．(それがどうした)

この章では，オントロジー構造と知識ベース構造を定義した．この定義は 3 章で出てくるオントロジー工学のアプローチや階層化表現の基礎となる．また 4 章で出てくる，データ処理技術やアルゴリズムといったオントロジー学習の枠組みもこの定義の上に成り立っている．

2.2 オントロジーの構築とそのアプリケーション

オントロジーを構築して実際に適用していくところを概観する．まず，オントロジーを分類する．あるタイプのオントロジーは特定のアプリケーションにしか適用されないから分類は大事．低コストでオントロジーを適用するためには，オントロジーの使われ方をよく理解しておく必要がある．

オントロジーの分類　Guarino は概念化の目的別に分類することを提案し，一般性のレベルによって図 2.4 のように分類した．

Top-Level ontology 時間・空間・イベントといった，特定の問題や対象領域から独立した非常に一般的な概念を記述．これはさまざまなユーザーが共有することができる最上位レベル．

Domain ontology Top-level に含まれる概念を限定することで一般的な対象領域に関する語彙を記述する．

Task ontology Top-level のオントロジーを限定することで一般的なタスクや活動に関する語彙を記述する．

Application ontology 最も限定的なオントロジー．概念は，ある対象領域の実体が特定の活動をすることで果たす役割にほぼ対応する．

Heijst の分類では，概念化された構造の量とタイプによって 1) 述語オントロジー，2) 情報オントロジー，3) 知識モデルオントロジーの 3 つに分類する．

Omelyenko はオントロジー学習に特化した分析とそれに伴う学習法による分類を提案している．そこでは自然言語オントロジー (NLO) とドメインオントロジーを分類し，さらにオントロジーを学習することを第 3 の分類基準として考えている．

オントロジーベースアプリケーションの例　対象領域に特化したオントロジーを用いた概念的背景知識の記述の上に成り立っているアプリケーション．

- ビジネスプロセスモデリング，タスク支援
- デジタルライブラリ
- マルチデータベースシステム
- WordNet（シソーラスのひとつ）を用いた情報検索
- 情報統合
- 知的エージェント
- 機械学習，データマイニング，テキストマイニング
- マンマシンインタフェース

さらに 4 つの主なアプリケーションについて詳しい事例をあげる．

セマンティックウェブ　WWW の技術は枯れてきていて，ウェブのソースから人間に情報を提供したり機械に知識を伝えたりできるようになっている．セマンティックウェブは，意味の形式的記述に基づいた自動化サービスをできるようになるべきだ．広がり続けるウェブ空間を横断していく道筋を見つけるとき，いまは文字列の一致をとることしかできないが，これからは意味がその鍵となる．

セマンティックウェブは，形式的オントロジーに強く依存する．形式的オントロジーは，計算機理解を目的とした包括的基礎かつ移転可能なデータを構造化し，データやメタデータをきちんと定義

することができる．一般にはセマンティックウェブはアプリケーションというよりはひとつのビジョンだという考え方もある．

図 2.5 は,「オントロジー」と「関係メタデータ」について示している．これは SWRC(Semantic Web Research Community) の一部で，まず Sigfried（ジークフリート）と　Steffen（ステファン）のホームページがある．これには RDF(Resource Description Framework) に基づく XML による注釈がついている．二人にはそれぞれ URI(Uniformed Resource Indicator) person_sha と person_sst として記述された実体がある．person_sha の SWRC:NAME は ”Siegfried Handschuh” になっている．また実体化された関数 cooperatedWith が二人の間にある．

他に，アプリケーションのシナリオに応じた知識ポータルといったセマンティックウェブアプリケーションもある．セマンティック Web の最新情報は，www.semanticweb.org からどうぞ．

自然言語理解　自然言語を包括的に理解するためには，多くの知識を統合しなくてはならない．オントロジーに記述されている対象領域に関する知識は，文章を深く理解するためには重要．特に概念的制約を満たしているかの確認や，読むべきところを選びだしたり，相互参照の一貫性の確認をするには重要．オントロジーで記述された知識を使って文章から知識ベースを構築する試みもある．機械翻訳にオントロジーを使う話もある．自由記述テキストから情報を抽出するためにオントロジーが果たす重要な役割というのは昔から言われていて，現実のデータにもそろそろ適用されつつある．

知識マネジメント　知識マネジメントが扱うのは，組織内での知識の獲得・管理・参照である．セマンティックウェブの技術によって，文書指向の見方から知識ピース指向の見方への転換が起こった．知識ピースは柔軟性の高い状態で相互接続されている．知的なプッシュサービスであるとか，知識マネジメントとビジネスプロセスを統合するであるとか，いつでもどこでも知識が得られるようにするための概念と方法が緊急に求められている．オントロジーはその鍵となる．オントロジーは，構造化されていない情報を意味情報とともに記述するために使われ，情報を統合して知識を参照しやすくするためにユーザに応じたビューを生成するためにも使われる．

e-ビジネス　商取引を自動化しようとするとき，商品について単なる字面だけではないフォーマルな記述が必要となる．相互運用を可能にするためには，用語とその翻訳についての共通理解をオントロジーとして記述しておく必要がある．そして知的な情報統合をするため，標準語彙の必要性も最近明らかになってきた．そういった語彙は文法的な記述の域をこえないが，オントロジーを定義するための起点としては適当だ．

2.3　結論

この章では，計算機科学がなぜオントロジーの「概念」を必要とするのかについて述べてきた．まず哲学におけるオントロジーの起源について述べ，意味の三角形に基づくパラダイムについて述べた．またどのようにオントロジーが人間と機械のコミュニケーションを支援するかについても示した．意味の三角形に流れるアイデアがオントロジーの「記号論的視点」と結合され，オントロジー構造と知識ベース構造へとつながっている．これらの構造の核となる要素とそれらの関係について述べ，フォーマルな定義を行った．先に述べたようにこの階層構造はこの本全体に関わっている．次章ではそれらの構造を実体化させる階層化フレームについて述べる．

Chapter3: LAYERD ONTOLOGY ENGINEERING

M2 身野 良寛

2002 年 8 月 25 日

近年、オントロジー工学が発展し、人工知能の研究からドメインの専門家にまで広がっている。オントロジーは Web において一般的なものとなり、小規模なものにはサイトのカテゴリー化 (Yahoo!)、電子商取引 (B2C,B2B) における商品分類、大規模なものには knowledge portal 、自然言語理解システムなどがある。

ドメインに特化した多数のオントロジーによって、Semantic Web が作られるが、以下の 2 つの理由により、オントロジー工学における新しい手法が必要とされている。

- オントロジーの専門家だけでなく、ドメインの専門家もオントロジーをモデル化・維持管理する。
- アプリケーションによっては、人間もコンピュータもオントロジーを利用するので、新しいオントロジー工学のパラダイムが必要である。

オントロジーの手法、表現形式、モデリングツールが開発されているが、それらは Semantic Web を想定して開発されたものではない。そこで、本章では 1 節でオントロジーの表現や処理をするために、既存研究を拡張し、Semantic Web のためのオントロジー工学のフレームワークを提示する。さらにこのフレームワークに基づいて開発されたオントロジーとアプリケーションの実例によって、異なる種類のオントロジーが開発される手順を示す。このフレームワークは 2.1 と 2.3 で定義した階層的なオントロジー構造 $\mathcal{O}$ と 知識ベース $\mathcal{KB}$ に基づいており、7 章で述べる OntoEdit の基盤となっている。

オントロジーの表現形式は Semantic Web のために WWW consortium W3C が定義した Resource Description Framework (RDF) に従っている。2 節では RDF にある階層を使って、どのようにして特定のオントロジー記述言語、ドメイン、知識ベースを表現するかについて説明する。そして、formal semantics(形式的意味論) を構成するために、表現形式のプリミティブを logical layer にマッピングする方法を説明する。最後に意味論的パターン (semantic pattern) を紹介し、それを論理言語へマッピングする方法と推論エンジンで処理する方法を示す。

3 節で結論を述べて、マルチユーザーやオントロジーの結合・発展に関する問題を提起する。この結論は 4 章への準備となる。

1 オントロジー工学のフレームワーク

すでに述べたように、オントロジー工学のフレームワークは $\mathcal{O}$ と $\mathcal{KB}$ に基づいている。本節ではオントロジーのインクリメンタルで循環的な開発について紹介する。オントロジーベースのシステムを構築する包括的な手法に関する議論は継続中である。ここで述べる手法を利用すると、2 つの問題に取り組むことができる。

- オントロジー工学において formal semantics が重要であるが、計算機と人間がオントロジーを交換するために、lexical references の特殊な機能を考慮しなければならない。
- 2 章ではオントロジーと知識ベースを別々に定義したが、実際には両者を厳密に区別しない。

図 3.1 は階層的で循環的なオントロジー工学の概略図である。左側は Ontology のプリミティブで右側は knowledge base のプリミティブである。

最下位の層と中間の層は図のように 2 章で定義したプリミティブからなる。最上位の層で定義される $\mathcal{A}^{\mathcal{O}}$ はオントロジー公理の集合である。

インクリメンタルなモデルは主に 2 つのインタラクションで構成される。

縦のインタラクション 階層の間の依存関係

ドメイン依存の公理を定義するとき、コンセプトの定義が必要

横のインタラクション オントロジーと知識ベースの依存・包含関係

デフォルトインスタンスの定義 (観光旅行ドメインにおけるコンセプト"CITY")

一般にオントロジーエンジニアは字句エントリ収集から始める。続いて一方ではコンセプト、関係、公理を定義し、他方では、特定のアプリケーションを想定してオントロジーを構築する。よって、知識ベースのエントリーはオントロジー構築の際、すでに定義されていることがある。これらのエントリーはオントロジーのデバッグと、アプリケーションにおけるエントリーの制約を与えるために定義される。(例、人材について考えると、知識ベースは従業員をインスタンスに含んでいる)

本節で示すフレームワークは 7 章の OntoEdit で実装されている。

オントロジー工学の例

1. SEAL-II
   knowledge portal の構築。アプリケーションは人材関連のドメインポータルをなし、Web 巡回による関連ドキュメントの収集をサポートする。階層的手法を使うと、まず人材関連の字句エントリ (労働時間モデル、E-Learning など) が収集される。コンセプト集合 $\mathcal{H}^{\mathcal{C}}$ は字句エントリを一般化することによって得られる。後に、ユーザー間で $\mathcal{R}$ について議論する。(例、TOPIC と PERSON の関係 CONTACT)

2. GETESS
   自然言語によるクエリや Web ドキュメントの要約処理をサポートするサーチエンジンの構築。自然言語処理において字句エントリが発生する。また、ユーザーインターフェイスに現れる字句エントリがコンセプトと関係にマッピングされる。このように階層的オントロジーによって、自然言語処理と同じようにユーザーインターフェイスのコンポーネントが提供される。

2 階層的オントロジーと知識ベースの表現のためのアーキテクチャ

本節では計算機で処理するための、オントロジーと知識ベースの表現形式に焦点を当て、共通で利用できる情報モデルと言語アーキテクチャを示す。

図 3.2 は階層的な言語構造を表している。

- syntax layer
  Extensible Markup Language(XML) を使った記号的表記

- data layer
  Resource Description Framework(RDF) による計算機が処理可能なデータ形式。RDF は XML に従ってデータモデルを表現する言語である。

- representation vocabulary data
  異種の”language primitive vocabulary” 我々は RDF-Schema が表現言語として最小共通分母であると考えている。RDF-S 仕様はセマンティックネットに非常によく似ている。この層で特定の知識や具体的な操作に関するボキャブラリを追加定義する。重要なのは、アプリケーションによってさまざまなボキャブラリ (OIL,DAML+OIL,DRDF など) が使われていることである。

2.1 RDF に基づいた Data Layer

RDF は要素間の関係を定義する抽象データモデルである。RDF におけるステートメントは人間や組織のような実世界の物体の代理や Web ページ (resource) を表現する。それを視覚化すると、関係のソースが主部、ラベル付き矢印が述部 (属性)、デスティネーションが目的部となる。

RDF データモデルはリソースとリテラルを区別する。関係はリソース同士、またはリソースからリテラルに向いている。関係自身もまたリソースであり、ステートメントに関するステートメントを作ることができる。図 3.3 は次のようなステートメントを RDF で表現したものである。

1. William is married with Susan.
   URI で記されたリソース同士が marriedWith という属性で関連付けられている。

2. The marrage between Willam and Susan has been confirmed by the resource.

3. 図の (c) は予め定義されたリソース rdf:statement と属性 rdf:subject,rdf:predicate,rdf:object を利用してリソースとしてのステートメントを図示したものである。(c) は (b) の実際のデータモデルである。ステートメントは真である必要はないことに注意すること。

2.2 Representation Vocabrary Layer

この層では異なる”labguage primitive vocabularies” を提供する。RDF-Schema が表現言語の最小共通分母であることにも言及する。以下、RDF-Schema がオントロジーと知識ベースの核心を表現することを示し、追加ボキャブラリに対する RDF(S) の拡張について述べる。

RDF-Schema RDF-Schema は RDF にタイプを追加するシステムを導入したものである。例えば、RDF-S はコンセプト (クラス) 階層や属性の適用範囲を定義する手段を提供している。図 3.4 にオントロジーが抽象データモデルにおいて、どのようにモデル化されるのかを例示する。

- 最も一般的なクラスは rdf:Resource である。このクラスは図のように 2 つのサブクラスを持つ。RDF をドメインに特化させるとき、ここでクラスや属性を定義する。これらは 2 つのサブクラスのインスタンスとなる。

- rdfs:Class,rdfs:Property はオブジェクト指向の意味においては全てのクラスの集合である。
- rdfs:subClassOf,rdfs:subPropertyOf は階層構造を表す特殊な属性である。
- RDF-S は属性のドメインや適用範囲を制限することができる。例えば、MarriedWith は Person に適用される。

underlying syntax　RDF で表現されたデータと XML は容易に相互変換できる。以下では RDF モデルを表記する XML の概略を説明する。

The use of XML Namespace in RDF(S)　XML 名前空間が RDF の階層的表現において重要な役割を果たす。名前空間は存在するスキーマやアプリケーションを再利用・収集するために、モデルの階層を識別する。

図 3.5 に XML を例示する。XML による RDF の表記は、参照に利用される ID の記述から始まる。次に、属性とリソースを入れ子の内部に記述する。

図 3.6 はこのオントロジーに従って作られたインスタンスの例である。app:Man,app:Woman クラスは URI を ID としてインスタンス生成されている。app:Man クラスは 2 つのリテラル値と 1 つの属性を持つ。

Extension of RDF-Schema with Representation Primitives　図 3.2 にあるように、RDF の上部にもう一つ階層がある。オントロジー工学で使われる lexical layer を追加する例を図 3.7 に示す。lexical layer のインスタンス生成を行なうために、新しい名前空間上に新しいプリミティブを定義する。これは図 3.5 のオントロジーの記述に類似している。図 3.8 はこのインスタンス (Organization の字句エントリ Organisation) を記したものである。そのボキャブラリを利用するとき、特殊なクラス (o:LexicalEntry) を参照する。

2.3　Logical Layer - Mapping to Formal Semantics

RDF(S) は formal semantics でオントロジーを記述する言語のスタート地点に過ぎない。近年、RDF を論理に基づいて形式化することやセマンティクスを明確にすることが試みられている。以下にそのいくつかの取り組みの概略について述べる。

- “Simple Logic-based RDF Interpreter(SiLRI) は RDF 3 つ組を F-Logic に変換し、そこに信頼できる推論システムを適用するものである。
- F-Logic セマンティクスを基にして RDF セマンティクスを記述する。RDF(S) プリミティブは直接、対応する F-Logic にマッピングされる。
  ((Frank,worksWith,Ole) は `Frank`[`worksWith` ↠ `Ole`] にマッピングされる。)
- 1 階論理 で RDF セマンティクスを捉える。この論理は SiLRI に受理される。
- DARPA Agent Markup Language(DAML) はセマンティックウェブに関する取り組みを取りまとめようとするものである。RDF,RDF-S,DAML を DAML+OIL というボキャブラリによる 1 階述語計算によって公理化することが提案されている。これはマッピングによって得られる論理は同じ意味を持っているということを主張している。伝統的な自動定理証明システ

ムを使って、この記述から推論を生成することが可能である。公理は Knowledge Interchange Format(KIF) で記述される。

これから、オントロジーと知識ベースが semantic patterns を使ってどのように F-Logic にマッピングされるかを具体的に述べる。

Semantic Patterns RDF-S は representation layer における ボキャブラリであると述べた。RDF-S はオントロジーを表記するには不十分であるので、logical layer を表記するための様々な拡張言語が提案されている。それ故、異種の言語やモデルを経由しても再利用できる方法を持ち、計算機処理可能な knowledge が開発された。これは RDF と semantic patterns を拡張したものである。semantic patterns はセマンティックウェブ開発者どうしのコミュニケーションにも、言語間のマッピングや再利用にも用いられる。semantic patterns の開発手法は、セマンティックウェブで利用するために、ソフトウェア工学や知識表現の研究を応用したものになっている。図 3.2 によれば、一般に semantic patterns は representaion layer と logical layer を接続するものであると考えられる。

The core idea semantic pattern に関する研究はモデルを抽象化しようとする公理図式に動機付けられた。公理図式は 1 種類の言語しか考えていないので、新しい認識論的プリミティブを導入することにより、異種言語で記述されたモデルのインスタンス生成が可能になる。どんな言語にも完全に変換可能なセマンティクスの仕様は見出されていないので、セマンティックウェブの開発者による連携が必要となった。これはソフトウェアエンジニアがソフトウェアデザインについて繰り返し討論することに類似している。

オブジェクト指向のソフトウェア開発のためにデザインパターンというものが考案された。その目的は、共通のボキャブラリを使ってドキュメンテーションと再編成可能なソフトウェア開発を容易にすることである。デザインパターンはソフトウェアエンジニアの間でデザイン案を伝えたり文書化するために用いられる。

表現形式の間の橋渡しは、型にはまった作業のように思われる。しかし、セマンティックスのすべてではなく、関連のある部分だけが変換される必要がある場合、そうではないことが多い。そのため、semantic pattern は形式的に定義される新しい認識論的プリミティブだけでなく、デザインパターンにあるような非形式的な伝達手段を含んでいる。

Example 図 3.10 は RDF(S) における公理のモデル化の略図で、RDF(S) の定義と semantic pattern のための拡張が描かれたものである。

次に示す逆公理は、働いていない参加者がいるようなプロジェクトは存在しないという一貫性を保証する。

$$\texttt{FORALL}\ X, Y \quad X : Person[\texttt{WORKSIN} \twoheadrightarrow Y] \leftarrow Y : Project[\texttt{PARTICIPANT} \twoheadrightarrow \texttt{X}]$$

この公理はたとえば逆関係の semantic pattern を使えば、

$$\texttt{INVERSEREL}(\texttt{WORKSIN}, \texttt{PROJECT}, \texttt{PARTICIPANT}, \texttt{PERSON})$$

と記述できる。この記述は簡単に F-Logic に変換できるという利点がある。

次の例は X が Y の父で Y が Z と結婚していれば、 X は Z の義父であるという公理である。

$$\texttt{FORALL}\ X, Y, X \quad X[\texttt{FATHERINLAWOF} \twoheadrightarrow Z] \leftarrow X[\texttt{FATHEROF} \twoheadrightarrow Y]\ \texttt{and}\ Y[\texttt{MARRIEDWITH} \twoheadrightarrow Z]$$

COMPOSITION(FATHERINLAWOF, MARRIEDWITH, FATHEROF)

この Composition という関係は RDF(S) で (5) のようにインスタンシエーションされる。F-Logic への変換は (6) のように直線的に行われる。

The Ontology and Knowledge Base structure オントロジーと知識ベースは、上のように簡単に記述言語に変換できる semantic patterns と考えられる。同様にすれば、表のようにオントロジーと知識ベースから F-Logic の公理へマッピングすることができる。

ここで、我々はマッピング可能なプリミティブを 4 つに制限する。semantic pattern のライブラリを作成すれば、プリミティブ $\mathcal{A}^{\mathcal{O}}$ への拡張も可能である。

Pattern Libraries どの種類の推論システムをサポートするのか、どの表記方法と変換可能にするのかということに関する新しいアイデアが生まれ、広く認知されるような semantic pattern が集められている。

- Gruber's Frame Ontology には 60 以上のプリミティブ集合がある。(対称関係、合成など)
- 医学における部分全体問題。
- 例外つきの継承には semantic pattern が有用である。例えば、患者の治療は医療保険が適用され、従順でない患者の治療には適用されない。しかし、精神病患者の治療には適用される。この場合、保険の適用範囲は従順でない患者を除いて患者のサブクラスに継承される。
- 局所的な通用範囲。"parentOfHuman" は人間に限定されるが、"parentOf" は動物一般に通用する。

3 結論とオントロジー学習への準備

本章では階層的オントロジー工学について紹介した。RDF(S) による階層的な表現言語を定義し、具体的にオントロジーと知識ベースを構築した。さらに、形式的意味論と記述言語との間のマッピングを紹介した。

semantic patterns はセマンティックウェブ開発者間の伝達に使われ、異なる表記間のマッピングを行うために用いられる。今後もこれに関する研究が続けられると思われる。特に semantic patterns のライブラリ発展に伴うコード生成に関して、ソフトウェア工学的手法がどのようにもたらされるかということが研究されるだろう。

本節では本章であまり触れられなかった協調とマルチユーザ、オントロジーと知識ベースの統合について触れておく。最後に、半自動的な方法でオントロジーを発展させる学習のための必要事項を定義する。

3.1 オントロジー工学関連のトピックス

セマンティックウェブのためのオントロジーを実現するために研究されるべき課題を 3 つ提示する。

Cooperative & Multi-user Ontology Engineering　オントロジーは知識共有と分散人工知能の研究に由来するものであるため、開発プロセスは複数の人物によって共同で行われると考えられてきた。特にオントロジーが受理されるためには、共有のボキャブラリを作成することが重要である。よって、開発・標準化プロセスに参加するユーザが増えれば増えるほど、関心あるドメインをより良く被覆することになるだろう。

クライアントサーバーモデルによる協調的オントロジー工学は複雑であるために、あまり深く研究されていない。形式的意味論における一貫性とマルチユーザをサポートするためにはデータベースにおけるトランザクション管理にのようなコンセプトが必要であるが、これもまた複雑である。

Multiple and Federated Ontologies & Knowledge Bases　中心的オントロジーによってオントロジーベースのウェブコミュニティがうまく機能している。しかし、グローバルなカテゴリー化が問題となっている。中心オントロジーの形式では、すべてのコミュニティのメンバーがそれを同じように受理するとは限らない。実際、コミュニティのメンバーは自身の観点を維持したいものである。そのような観点を記述するために、オントロジーマッピングによって明確な知識が交換されるようにオントロジーを分散させる必要がある。ユーザによってカテゴリー化されたものを分析して、自動的にオントロジーマッピングを行う方法がある。これは次章で説明するオントロジーマージングである。

オントロジーを統合する手法はマルチデータベースシステムを基にしている。また多重シソーラスデータベースにおいても、分散したシソーラスデータベースを統合する方法が紹介されている。

この分野には関連する研究が数多く存在するが、オントロジーと知識ベースの共同利用可能性に関しては問題が多い。(マッピングとマージングの管理、同一性の識別など) オントロジーの維持管理に関する問題は 6 章で述べる。

3.2　オントロジー工学のためのオントロジー学習

オントロジー構築、記述言語、オントロジー工学のツールはこの 10 年間で成長したものである。しかし、人手による知識獲得やオントロジーモデリングが厄介な作業であり、ボトルネックとなっている。

SEAL,SEAL-II のような知識ポータル開発によって、この問題に取り組んでいる。我々が特に受ける質問は次のようなものである。

- どのくらいの早さでオントロジーを構築できるのか。
- オントロジー構築は難しいのか。どうすればオントロジーを使ってドメイン知識が明示されるのか。
- どのようにしてオントロジーがあるドメインを良く被覆しているということがわかるのか。
- どのようにしてオントロジーを維持するのか。

実際、これらの時間・難しさ・信頼性・維持に関する問題は知識工学が扱ってきたものに類似している。

知識工学において非常に有用な方法は機械学習の技術で知識を獲得することであった。現在機械学習は多少複雑な知識ベースのインスタンスに限定されている。

WWW は様々なデータを利用する。そのため、WWW ではオントロジーの構築を容易にするために多数の学問を統合しようとする。オントロジー学習を完全に自動化することは遠い将来の課

題であるので、学習プロセスは人間が干渉する半自動的なものであると考えられている。これを念頭において、Web にあるリソースを使って知識獲得と機械学習を結びつけるアーキテクチャが作られた。したがって、手動でも可能なオントロジー学習を統合する方法を考えてみると、満たされるべき要求が多数存在する。最後に、この要求を定義する。

- オントロジー構築の各ステップは手動または自動で処理される。オントロジー学習はいつでも有効・無効を切り替えられるプラグインとみなされる。また、アルゴリズムは $\mathcal{O}$ にある関連するすべての要素に対して利用可能であるべきである。
- オントロジー学習のフレームワークはオントロジーエンジニアに操作されたいフェーズを定義すべきである。
- フレームワークはユーザーインタラクションの方法を提供すべきである。
- オントロジー学習のために、ユーザが関連のあるデータを探すとき、ガイドを行うべきである。システムによってデータの処理が行われ、異種のデータの結合が可能でなければならない。
- 既存のオントロジーの再利用がサポートされるべきである。目的のドメインに対して複数のオントロジーが利用可能であれば、ドメインが重複したオントロジーを結合・マージする手段を提供しなければならない。
- 既存のモデルやオントロジー構造 (階層的コンセプト $\mathcal{H}^{\mathcal{C}}$) は、アルゴリズムの内部において背景知識として利用され、またモデルの増分や改良をサポートすべきである。
- アルゴリズムは最適なものが 1 つ存在するというわけではないので、同じ問題を解決するアルゴリズムを複数使うことができるようにすべきである。そして、それらは結びつけられなければならない。
- フレームワークは結果の表示や複数結果の結合を表示する方法を提供しなければならない。特に、発見された提案の数を制限する手段に関する重要な疑問に取り組まなければならない。

次章ではオントロジーの抽出や維持を行う半自動な手段をサポートするための、一般的なフレームワークを提示する。

Ontology and the semantice web

亀田能成
kameda@media.kyoto-u.ac.jp
2002/08/24-27, version 1.5

Chapter 4 : A Framework for Ontology Learning

Semantic Web のための総合的なオントロジー学習の枠組みを紹介する。この枠組みは、オントロジーの基盤 (foundation) と定義、knowledge base structure、そして後の２章で紹介される階層オントロジー工学のパラダイムに基づく。

ここでは、コンポーネント、コンポーネント間の相互作用 (interaction)、それに、「オントロジーの抽出 (Extraction) と維持」に対し実世界データオントロジー学習を適応する上で必要な全ての処理を紹介する。

本章で述べる枠組みは本書の**屋台骨**を成す。本章の構成は三部からなる。

第一部では、オントロジー学習のための適切な (relevant) 入力になるとされる様々な種類の **data** について述べる。relevant data の分類 (Classification) は様々な型のデータとそれらの間の関係を導入するために使われた分類学 (Taxonomy) の形式で表現される。データの様々な型の特別な特徴も紹介する。

第二部では、オントロジー学習に基本的に必要だと見なされている以下の４主要コンポーネントについて述べる。

1. オントロジー工学とマネージメント環境 (The Ontology Engineering & Management Environment)
2. データ取込（インポート）と処理コンポーネント (the Data Import & Processing Component)
3. アルゴリズムライブラリコンポーネント (the Algorithm Library Component)
4. GUI とマネージメントコンポーネント (the Graphical User Interface and Management Component)

これらは **comprehensive architecture** 内へ埋め込まれていく。これらのコンポーネント間の相互作用の解析が重要である。これについては第２部で詳細に述べる。続く５章では data import & processing について、６章では ontology learning algorithm について詳細を述べる。ここで述べる Comprehensive architecture は、TEXT-TO-ONTO という名前のオントロジー学習工学環境 (Maedche and Volz, 2001) で実装されている。これについては７章参照。

第３部ではオントロジー学習における **proces-oriented view** を紹介する。これは主に４フェーズからなり、**import, extract, prune, and refine** である。これらは **ontology learning cycle** を成す。これらの４フェーズに基づいて、オントロジー学習を「オントロジーの抽出と維持 (extraction and maintenance of ontologies)」に適用する道筋について明らかにする。

The Semantic Web における一つの重要な側面は、現存する知識構造を取り込み (import)、新しい構造を抽出 (extract) することでオントロジーを作り上げていくことではなく、ある an ontology の刈

り込み (prune) をしたり、その the ontologyn の改良をするのに必要な「維持フェーズ (maintenance phase)」を考えることである。

1 A Taxonomy of Relevant Data for Onotology Learning / オントロジー学習のための該当データの分類

オントロジー学習アルゴリズムを適用する前に、データの様々な型其々には、特別な import / processing / transformation を施す必要があると読者は気づくであろう。これについては５章。

Fig4.1 が relevant data の分類を示す。主要型は５つありそれらは、ontology, schemata, instances, web documents, semi-structured data である。

Data in the form of ontologies （存在論） Fig4.1 の左枝参照。オントロジーはデータの一つの特別な種類として考えられうる。

オントロジーのいろんな種類としては、WordNet やそのドイツ語版の GermaNet のような linguistic ontologies や, thesauri, domain-specific web ontologies などがある。

また、すでに存在するオントロジーの再利用 **reusing** することは重要である。

The Semantic Web はオントロジーの形式でもって domain-specific schemata の上に構築される。

本書では、再利用可能オントロジー構造は様々な種類のオントロジーにも利用可能であると考える。例えば、WordNet や GermaNet のように、大規模で言語的な **lexical-semantic nets** がそうである。また、観光のオントロジーから金融のオントロジーへの切替が GETESS で行われたのは **domain-specific onotologies** の例である。**thesauri** は軽量なオントロジーで、より総合的なオントロジーを導出するために再利用されうる。

現存するオントロジーや類似するオントロジーをどのように取り込む (import) かについては、次章で議論する。

Data in the form of schemata （概要） 情報システムはもっぱら (semantic) データモデルに依存する。情報システムは entity-relationship model や object-oriented model として与えられたりする。

こうした「Modeling paradigms and mechanisms」は世界のアプリケーション依存部分を概念的に表現するために開発されてきた。Schemata という種類に含まれる工学化 (Engineering)・モデル化 (Modeling) の努力は、the Semantic Web に対するオントロジー学習のために再利用されるかもしれないということは明白のように思える。

これからちょっとだけオントロジー学習にまつわる schemata について述べるが、本書ではそれ以降は深入りしない。というのも本書の焦点は The Web 上の semi-structured and free text documents であるからである。次章で述べるオントロジー学習アーキテクチャは一般的な話なので、取込 (import)・現存する schemata のリバースエンジニアリングなどを導入することは簡単であることに注意されたい。

- Database Schemata

  モデリングという意味では、(Chen,1976) によって導入された **entity-relationship model** が古典的技法の一つ。

典型的なところでは、Lang and Lockemann が 1995 に entity-relationship model から **relational database schemata** を生成している。

現存する schemata の再利用工学 (re-engineering) と調停サービス (mediation service) に関する問題は長年研究されてきている。しかし、source schemata から mediated schemata へのマッピングを構築するのは大変な人手作業でそこがボトルネックとなっている。

- Web Schemata

  最近はやりである。ここに出てくる不全構造データ (semi-structured data) には、なんらかの構造が隠されているにしても固定・厳密な schema がないのが問題である。XML(eXtensible Markup Language) の DTD(Document Type Definition) や XML-schema も Web Schemata に相当しよう。これまではあまり研究されてこなかったが、DTD からオントロジーを導出する研究も行われてはいる。

Data in the form of instances （事例）　事例 (instance) とは外延的 (extensionally) に定義されたオブジェクトと考えられる。知識ベース構造 $calKB$ に 基づいて与えられたオントロジーの上に成り立つ定義 2.3 参照のこと。データや知識ベース内の事例を集めれば、ドメインコンセプトを外延的に表現することになる。つまり、あるドメインに対する恣意的な表現を帰納的に抽出するために、関連するデータを事例らは表現していることになる。

事例から学習をする研究は、**concept formation or conceptual clustering** と呼ばれる。最近の研究では **A-Box-Mining** が有名らしい。

Data in the form of semi-structured data （"不全" 構造データ）　前にも述べたように、不全構造データ (semi-structured data) には、なんらかの構造が隠されているにしても固定・厳密な schema がないのが問題である。そのため、不全構造データの抽出は簡単でも、それを表示したり検索 (querying) したりするのは大変困難なこととなる。ゆえに、不全構造データに埋まっている構造をいかに発見して書き直すかがこの問題への鍵となる。

Data in the form of natural language (NL) text　自然言語の文章は WWW 上で大量・無料に存在するので、これは**オントロジー学習のための入力データとしてもっとも重要**と考えられている。

- Pure natural language text
  自然言語を相手にするとき、言語学的制約がオントロジー学習への重要な手がかりとなる。詳細は 5 章にて。

- Natural language documents enriched with semi-structured information
  最近では HTML データなどには大なり小なり意味的情報が含まれていたりするので、WWW 上のデータはだいたいこの範疇に属する。

Conclusion　以後の節ではオントロジーと自然言語文章に限って話を進める。

2 An Architecture for Ontology Learning / オントロジー学習のためのアーキテクチャ

ここではオントロジー学習アーキテクチャとそれに関連するコンポーネントについて紹介する。オントロジー学習のフレームワークに必要なことは、５つに集約される[1]。

2.1 Overview of the Architecture Components / アーキテクチャコンポーネントの概観

アーキテクチャの中には下記の４主要コンポーネントが組み込まれなければならない。

1. Ontology Engineering & Management Environment “ONTOEDIT”
   ONTOEDIT は統合オントロジー管理機能と GUI を提供し、オントロジーエンジニアが手動で行うオントロジー工学処理をサポートする。ONTOEDIT は、３章で述べた階層オントロジー工学がもとになっている。詳細は７章。

2. Data Import & Processing Component
   本コンポーネントには、関連データについて発見 (discovering)・取込 (importing)・解析 (analyzing)・変換 (transforming) を行うための様々な技術が含まれている。重要なのは自然言語処理システム（図 4.2 左上）である。内部については次章参照。

3. Algorithm Library Component
   本コンポーネントは本フレームワークのアルゴリズム的バックボーンを成す。オントロジー構造 $\mathcal{O}$ に含まれるエレメントの抽出や維持を行う様々なアルゴリズムがある。どんな学習方法にも対応できるよう、共通結果構造 (common result structure) で結果は出力する。もしいくつかのアルゴリズムが同一結果を返す場合は、それらを融合し、ユーザにはただ一度だけそれを提示する。

4. Graphical User Interface and Management Component
   オントロジーエンジニアがデータ取込・処理やアルゴリズムライブラリを使用するときに、このコンポーネントを用いて対話処理を行う。７章で様々なインターフェースについて取り上げる。

図 4.2 にアーキテクチャ内の４コンポーネントの関係を図示する。**functional components**(データ取込・処理とアルゴリズムライブラリ) は **interface components**(GUI や ONTOEDIT) によって隠蔽されていることに注意する。

本アーキテクチャでは、全てのコンポーネントがエンジニアが作業している**オントロジーにアクセスする**ことになる。各コンポーネントはオントロジー内の予備 (background) 知識を利用する。自然言語処理モジュールは用語集 (lexicon) を通じてオントロジーモデルに接続している。

[1] ３章で述べられているらしいのでここでは略。

2.2 Ontology Engineering Workbench “ONTOEDIT” / オントロジー操作環境 “ONTOEDIT”

層状オントロジー操作において、各層は個別のモデリングやオントロジー表現を区別するために用いられる。ONTOEDIT は層状オントロジー操作環境を提供し、手動でモデル作りが可能で、GUI がついている。図 4.3 は動いている様子の写真である。図中では、特定の表現言語ではなく概念レベルを考えているユーザにオントロジーを様々な方法で見せている。図中左には概念階層 $\mathcal{H}^{\mathcal{C}}$ が表示されている。図中中央には、概念階層中のドメイン概念 $\mathcal{D}$ について、特定のレンジ $\mathcal{R}$ に関連する非分類学 (non-taxonomic) 関係が表示されている。図中右には、すべての関係 $\mathcal{R}$ が特定のドメイン・レンジ概念と独立に表示されている。下段右には、オントロジー公理 (axiom)$\mathcal{A}^{\mathcal{O}}$ の定義が表示されている。

ONTOEDIT の特徴のひとつは、OIL, DAML+OIL, F-Logic などの異なる表現言語にも対応可能であることである。新しいのが必要になれば、ONTOEDIT 依存なオントロジー表現プリミティブを OntoEdit namespace で表現すればよい。

このオントロジー学習アーキテクチャの利点の一つは、様々な表現言語でオントロジーにアクセス可能であることである。現在 ONTOEDIT は SiLRI F-Logic による推論エンジンになら直接アクセス可能である。このあたりのことも詳細は 7 章参照。

ここで述べた方法によって、高品質な Web のオントロジーを手動で生成可能であるが、こうしたオントロジー操作ツールと Web 文章のような入力との間にはまだ大きな **semantic gap** が存在する。以降ではこのギャップを埋め ONTOEDIT を拡張するための 3 つのコンポーネントについて述べる。

2.3 Data Import & Processing Component / データ取込・処理コンポーネント

本コンポーネントには当該入力データについて発見 (discovering)・取込 (importing)・解析 (analyzing)・変換 (transforming) を行うため様々な技術が含まれる。ここでは自然言語文章（自由文書、HTML、辞書等）と様々なオントロジーのみを取り上げる。本コンポーネントの重要なサブコンポーネントは**自然言語処理システム**である（図 4.2 左上）。

一般的には、本コンポーネントは、[1] アルゴリズムライブラリコンポーネントへの入力としての処理済 (preprocessed) データ集合か、[2] 内部オントロジーモデルの事例化 (instantiation) か、のどちらかを生成することになる。

本コンポーネントのほかのサブコンポーネントは以下のとおり。

1. Ontology Wrapper and Importer
   オントロジー取込を行う。また、取り込まれるオントロジー (WordNet, GermaNet) のための具体的表現言語依存「オントロジーラッパー」も提供する。

2. Ontology Mergin via FCA-MERGE
   二つ以上のオントロジーを取り込むときには、一方をもう一方に融合させることが必要になる。FCA-MERGE が使われたりする。

3. Ontology-based Focused Crawler
   オントロジーのある形式内にある予備知識を、Web 上の文書探索のために用いる。オントロジー学習のために関連入力データを集めるのを手助けするサブコンポーネントである。

4. Natural Language Processing System
   本書では、ドイツ語の簡単な文書処理器として SMES を用いる。

5. Document Wrapper
   辞書や他の不全構造文書を、アルゴリズム依存な目的とする関係記述に変換するためのサブコンポーネント。

6. Transformation Module
   言語学的に注釈付けられた文章（＝自然言語処理システムの出力）を、アルゴリズム依存な関係記述に変換するためのサブコンポーネント。

図 4.2 に示すように、本コンポーネントの最終産物は、[1] 事例化されたオントロジー構造か、[2] アルゴリズム依存な処理済データの「塊 (blocks)」か、のどちらかである。

2.4 Algorithm Library / アルゴリズムライブラリ

繰り返して言うが、オントロジー学習はオントロジー操作環境のプラグインであると考えられる。

エンジニアが最初は基礎的手法からスタートできるように、様々な対話処理の枠組みが提供される。予備 (background) 知識が不要ないしほぼ不要な手法がそれに該当するが、これらは簡単なヒント（単語の出現頻度など）しか出力してくれない。そこからスタートして徐々に複雑な手法が適用されていく（6 章参照)。

下記のように、大きく分けて 2 種類のアルゴリズムがある。

1. ontology extraction
   概念インジケータとしての単語エントリや、概念の分類学的順序、それにドメインの概念とレンジ制限 (range restrictions) との非分類学的関係などを抽出したりするアルゴリズムがこれに相当する。

2. ontology maintenance
   オントロジー進化・精製のためのオントロジー還元 (reduction)・刈込 (pruning) を行うアルゴリズムなど。

我々は「結果組み合わせ手法 (result combination approach)」を用いる。これは、複数のアルゴリズムで出てきたオントロジー構造をまとめて一つの形にする手法である。詳細については 6 章および 8 章。

2.5 Graphical User Interface and Management Component / GUI と管理コンポーネント

ONTOEDIT は GUI ベースになっているので、オントロジー学習フレームワークに本質的に含まれる技術的な難しさはすべて隠蔽されている。

1. 本 GUI 環境は関連データ (relevant data) の選択を手助けする。

2. パラメータ化を行ったり、オントロジー学習アルゴリズムをスタートするための様々なインターフェースを持つ。

3. 統合的な **result set views** を提供する。ここでは、結果を概観したり並べ替えたり選択したり対象となるオントロジーに付け加えたりすることができる。（７章に図示例あり。）異なる可視化アルゴリズムを用いて抽出結果を見ることができる機構も用意される。

次節ではオントロジー学習のためのアーキテクチャをプロセス指向で概観し、上記のコンポーネントが４つのフェーズにどのように埋め込まれるかを見ていく。

3 Phases of Ontology Learning — Import, Extract, Prune and Refine / オントロジー学習のフェーズ — 取込、抽出、枝刈、精製 —

プロセス指向でみると、オントロジー学習フレームワークは取込 (import)、抽出 (extract)、枝刈 (prune)、精製 (refine) の４つのフェーズからなる。図 4.4 参照。オントロジ―学習サイクルはブートストラップ機構に良く似ている。Jones et al の説明によれば、「オントロジ―学習アルゴリズムをまず種情報から初期化し、それから学習アルゴリズムをラベルのついてないデータのラベリングに繰返し適用し、こうしたラベルの幾らかを学習者のトレーニング入力にうまく組み入れていく。」

４つのフェーズは以下の通り。

1. 可能なら、まず現存するオントロジ―の取込と再利用から始める。本書では語彙意味的 (lexical-semantic) ネットをオントロジ―学習の入力として扱う。次章では、WordNet や GermaNet がいかに取込まれるかについて述べる。

2. ２段階目として、オントロジ―抽出フェーズでは、目的とするオントロジ―の大半が学習機構の支援を得てモデル化される。ここでは取込まれたオントロジーを予備知識として利用するかも知れないが、オントロジ―が事前に与えられてなくても（本フェーズは）機能するであろう。６章参照。

3. 第３段では、目的とするオントロジ―の大まかな概略が最終目標に向けて枝刈される必要がある。これについても６章参照。

4. 第４段では、与えられたドメインオントロジ―から便宜をうけて、オントロジ―精製を行い、オントロジーを仕上げる。

   このサイクルは新しいドメインをさらに導入したり維持／更新する場合、さらに繰り返される。これについても６章参照。

付け加えて言うならば、目的としているアプリケーションでもって結果のオントロジ―の有効性を測ることができる。

各フェーズは他のフェーズと独立して実行可能であるので、フェーズを飛ばしたりすることも考えられる。

これ以降は、Maedche and Staab, 2001 のオントロジ―学習を基礎にした話である。具体的な技術については５章および６章参照。

3.1 Import and Reuse / 取込と再利用

観光／遠隔通信／保険／金融での経験から、ある種のオントロジ―構造はほぼ全ての商業的重要なドメインに共通して利用可能であると期待できる。ゆえに、こうした構造の「取込／再利用」が必要となる。これは二つに分かれる。

- 関連するオントロジ―が選択され、取込まれる戦略が定義されなくてはならない。（１つの表現言語から別の表現言語への変換を支援する Ontology wrapper のようなものが定義されるべきである。）
- 取込／再利用段階では、複数の概念構造が融合される必要がある。これを元にして抽出／枝刈／精製が行われる。

融合 (merging) や整合 (aligning) はいまだ open problem であるが、手動で行われているこの作業を改善する研究も行われてきている。FCA-MERGE などもその一例であるが、これについては次章参照。

3.2 Extract / 抽出

関連する様々なデータを利用できるように、本フェーズではオントロジ―の大部分がモデル化される。

オントロジ―学習工学は与えらたオントロジ―の一部に部分的に依存したりする。

反復的ブートストラップモデルの事例が、６章に出てくる非分類学的関係の抽出技術である。概念間の関係は、コンセプト集合 $\mathcal{C}$ をもとにして抽出できるかもしれない。しかしながら、もしこれらの概念の分類学的順序 $\mathcal{H}^{\mathcal{C}}$ が与えられれば、より統合的で一般化された概念間関係が抽出できるかも知れない。このように、現存する予備知識を使えば結果の一般化が可能になる。詳細は６章参照。

3.3 Prune / 枝刈

モデル化するとき常に問題になるのは、完全さと不完全さのどちらを取るかという問題である。完全さを求めれば管理不能で計算量的にも手に終えなくなる。一方で、不完全さを認めれば表現力に制限がかかりすぎる[2]。

取込／再利用や抽出フェーズでは不完全（かつ膨大であまりに総合的な）モデルが生成されたりするので、サイズを押さえ込むことが重要である。そこで、二つの次元がここでは検討されるべきである。

- まず最初に、オントロジ―のどの部分を刈り込むと全体に影響を与えるかを、明確に知る必要がある。Peterson のや Swartout の手法は、ユーザには整合のとれたオントロジーを提供するものである。ONTOEDIT では、ユーザが特定の概念を削除しようとした場合、関連する全てのオントロジ―構造が計算されてユーザに呈示され、ユーザがその特定の概念が削除されるべきかどうかを決定する。

[2]他への転用とかが困難になるという意味か？

- 続いて、オントロジ―のアイテムが枝刈されるべきかどうかをどう呈示するかについて、戦略を考える。6章の2.1節にそうした機構の1例がでてくるが、そこでは関連文書の集合として事例化されない概念や関係は削除するべきだとしてユーザに提案する。

アプリケーション依存文書集合が与えられたとき、オントロジ―枝刈には幾つかの戦略がある。

- 単語出現頻度 (lexical entry frequencies) を計測する。出現頻度の低いものはオントロジ―から削除する。エントリについては、削除するか、ないしは、オントロジーに残っていくようなより一般的概念に近いものにマッピングし直す。

  単語出現頻度 (lexical entry frequencies) の代りに頻度／逆引文書頻度 (frequency / inverse document frequency, tfidf) を使うことも考えられる。

  付け加えて言うなら、コンセプト階層 $\mathcal{H}^C$ を用いれば分類学を用いてコンセプトを通した頻度を伝搬させられるかも知れない。

- 洗練された枝刈技術を用いれば、ドメイン依存に集めたの文章（例えば電化製品の記事）中の単語頻度を、一般的な参考文章（新聞の記事）中の頻度と比較できる。こうすれば興味深いが稀な頻度のエントリを枝刈りせずに済む。

上記2つめの話については詳細は6章にて。

3.4 Refine / 精製

精製は抽出と良く似ていて、明確な区別は難しい。抽出がオントロジ―全体のモデリングに関連するのに対して、精製は**目標とするオントロジーを洗練 (fine tuning) させること**と捉えられる。原則的にはどちらにも同じアルゴリズムが適用可能だが、精製のときには現存するオントロジ―とオントロジ―間の関係のことを吟味しなくてはいけない。

研究としては Harn and Schnattinger の例がある。

本書でのオントロジ―精製手法は、未知の単語エントリはすでに知られている単語エントリに対して類似した「概念的振る舞い」をすると仮定している。

4 Conclusion / 結論

英語で書いてあるという点を差し引いても、少なくとも4章あたりについては情報処理学会学会誌 2002 年 7 月号の特集のほうがわかりやすい気がするのは私だけでしょうか‥‥。

【おまけ】

本稿はセミナーでの議論を受けて修正を施した版である。以下に本章で初出でない単語について列挙する。

- lexicon / 語彙 (Chap 2)
- lexiccal entry / 語彙項目 (Chap 2)
- engineering / 工学
- instance / 事例（実体）(Chap 2)

Chapter 5
DATA IMPORT & PROCESSING

担当者：伊藤　淳子

2002/08/24-27

導入部分で述べたように，Semantic Web は実存する WWW 上に構築される Meta-Web と捉えることができる．Ontology は，たとえば実存する web 文書についてのメタデータの生成のためなど，例示化される語彙を表現することによって Semantic Web の生成をサポートするキー要素である．

前章では，Semantic Web に対する Ontology 学習のために我々のアーキテクチュアにどんな要素が含まれているかを示し，それらの要素がどのように相互に作用するかを示した．もし re-engineering process として Ontology 学習をみなした場合，新しい Ontology を導出するためにもとからあるデータ (legacy data) を取り扱うべきである．利用可能な元データや応用データは Ontology 学習の始めのポイントとして与えられる．従って，応用データと同様元データもソースとしてみなされる．

この章では，実存している web 上で利用できるデータがどのように探索され，アクセスされ，処理され，関係のあるアルゴリズム固有の表現へと変形されるかを示す．図 5.1 にこの章の概観を示す．この絵は図 4.1 で表現した関連データ分類を表している．読者は，この章では 2 つの特別な種類の関連入力データのみを扱っていることに気がつくかもしれない．その 2 つのデータとは **Ontology** と**自然言語文書**である．前に述べたように，実存する Ontology は新しいオントロジーを導出するための重要な知識のソースとして役立つ．実在する Ontology は，その枠組みの中でアクセスされるために変換され (解釈され)，融合される．この処理理論の最終結果が，Ontology 構造 $\mathcal{O}$ による Ontology である．文書に関しては，異なる入力と処理技術が Ontology 技術者に提供される．関連文書は Ontology に基づいた crawing アプローチによって発見され集められるだろう．この処理技術を適用して，domain-specific な”学習コーパス”になる．

本章の構成：

第 1 節　ontology import and processing に対する技術
まず，ある Ontology が明確な言語での表現で表されていたら，Ontology 学習の枠組みの中でその Ontology にアクセスするために，その Ontology は Ontology 構造 $\mathcal{O}$ へ包括される必要がある．2 つ目に，もし 2 つ以上の Ontology が取り込まれた場合，これらの Ontology を融合 (merge) するための技術は Ontology 技術者が提供しなければならない．

第 2 節　我々の Ontology 学習の枠組みの中で，文書を発見 (discovering)，利用 (accessing)，解析 (analyzing)，変換 (transforming) するための構造

第 3 節　本章をまとめ，これからの課題を定義

1　Importing & Processing Existing Omtologies

近い将来，より多くの Ontology が利用可能になり，あるドメインから別のドメインへの素早い適応や与えられた Ontology の拡張が重要になるであろう．前に述べたように，異なったソートをされた Ontology を識別する必要がある．たとえば，大規模で言語的な lexical-semantic net である WordNet やそのドイツ版の GermaNet，あるいは応用 Ontology(GETESS，German Text Exploitation そして Search System のような特別な応用のために作られた tourism ontology) などが異なるソートとして挙げられる．これらの異なったソートをされた Ontology は，非常に単純な表現から形式的な表現まで，表現言語に潜む，異なった複雑さをもっている．もし誰かがこれらの異なったソートをされた Ontology を取り込み処理したいなら，主として次の 2 つのステップを実行しなければならない．

- ▷ オントロジー学習の枠組みの中で使われうる表現 (representation) へと与えられた Ontology を変換しなければならない
- ▷ １つ以上の Ontology が利用可能なら，１つの共通 (common)Ontology へと融合されなければならない．

一般的に，Ontology import と processing に対して我々は公理 $\mathcal{A}^{\mathcal{O}}$ を考慮することなく，$\mathcal{O}$ に含まれている核心要素 (core element) に注意する．図 5.1 に示すように，２つのモジュールが実存する Ontology を取り込むために提供されている．その２つとは，オントロジーラッパーと Ontology 融合手法の FCA-MERGE であり，次の小節で説明する．

1.1 Ontology Wrapper & Import

Ontology には数多くの表現言語が存在する．web で広く使われるようになる以前から，すべての言語に対し１つの表現レベルを見出そうとしたり，自動的に別の言語に変換することなどに労力が払われてきた．どちらの方法も表現の意味，つまり意味的な含意が単一の共通語で充分に表現できないという事実に悩んできた．部分的にこの問題を取り扱う，*semantic patters* approach は第３章で簡単に述べたが，具体的応用の中で Ontology を使う際，次のような状況を経験してきた．特定のドメインや応用 Ontology が特別な表現言語で表現される場合，Ontology 構造 $\mathcal{O}$ による Ontology 固有表現を包みこむような wrapper を記述しなければならない．Ontology 構造の意味は，固有の写像によって定義される．

次に，lexical-semantic net がどのように枠組みに取り入れられるかを簡単な例で示す．

Importing WordNet/GermaNet

WordNet とそのドイツ語版は，lexical-semantic net であり，オンラインの語彙参照システムである．英語の名詞，動詞，形容詞，副詞などが同意語の集合に統合される．

どちらの lexical-semantic net も Ontology 学習には有用なソースである．どちらの net も Ontology 学習の枠組みの具体化された Ontology 構造 $\mathcal{O}$ 変換されてきた．次のリストは，この２つの net に含まれた，関係のあるかつ Ontology 的な基のリストである．

- ▷ **同意語 (SynSet)**
  同意語の集合．ある文脈で交換可能な言葉の集合．
- ▷ **上位語 (Hypernym)**
  特別な事実全体のクラスを指定するために使われる一般的な用語．X が Y(の一種) であるとき，Y は X の上位語である．
- ▷ **下位語 (Hyponym)**
  あるクラスのメンバを指定するために使われる特別な用語．X が Y(の一種) であるとき，X は Y の下位語である．
- ▷ **全体 (Holonym)**
  部分的な名前全体の名前．X が Y の１部であるとき，Y は X の全体である．
- ▷ **部分 (Meronym)**
  構成要素の名前であったり，何かのメンバである実体の名前．X が Y の１部であるとき，X は Y の部分である．
- ▷ **反意語 (Antonym)**
  共起性により作られる，関連のあるつながりがそれらの間にあるような１対の単語

表 5.1 は GermaNet と WordNet lexical-semantic ontology に対して実行される Ontology 構造 $\mathcal{O}$ への写像のリストである．

図 5.2 は左側にある WordNet と右側にある GermaNet から抽出された $\mathcal{H}^{\mathcal{C}}$ から抜粋した例を示している．

1.2 FCA-MERGE – Bottom-Up Ontology Merging

ontology wrapper approach は，与えられたドメインを利用できる Ontology がただ 1 つだけのときにうまく働く．しかし，たとえばお互いを補完するような 2 つのドメインから 2 つ以上の Ontology が取り入れられた場合，2 つの Ontology を 1 つの目標 Ontology にへと融合する手段が求められる．Ontology 融合の過程では，2 つ以上の Ontology を入力として得て，与えられたソース Ontology に基づいて融合された Ontology を返す．

FCA-MERGE と呼ばれる新しい手法は，融合処理の構造的表現を提供するような bottom-up の方法に従って Ontology を融合するために考え出された．この手法は融合される 2 つの Ontology の概念集合 $\mathcal{C}$ の拡張記述によって導かれる．概念の拡張記述は，自然言語処理システムを使って導出される．拡張記述は，いわゆるコンテキストへと保存される．このコンテキストは形式的概念解析 (formal concept analysis) と呼ばれる理論への入力となる．形式的概念解析は FCA-MERGE の構造的な結果として概念の枠 (lattice) を導出する．その結果はそれから探索され，人間とのやり取りのもとに融合された Ontology へと変換される．

1.2.1 A Short Introduction into Formal Concept Analysis

形式概念解析 (FCA:Formal Concept Analysis) の基本について述べる．拡張と目的からなる概念の数学的表記のために，3 つ組 $K := (G, M, I)$ として定義された形式的関係 (formal context) から FCA の説明を始める．ここで，G は事物 (object) の集合，M は属性 (attribute) の集合，I は G と M の間の 2 値関係 (すなわち $I \subseteq G \times M$) である．

定義 5.1 により，形式的概念が数学的に定義される．関係K の，partial order が $\leq$ であるような全ての形式的概念の集合が常に完全な枠である場合 (すなわち，各形式的概念の集合に対し，常に最大の subconcept と最小の superconcept が存在する)，この枠をK の概念の枠 (concept lattice) と呼び，$\underline{\mathcal{B}}(K)$ と表記する．

1.2.2 The FCA-MERGE Method

2 つの Ontology を FCA-MERGE を使って融合する処理は，以下の 3 段階からなる．

1. 拡張概念記述 (extensional concept description) の抽出と，2 つの形式的関係 K1 と K2 の評価
2. 一般的な関係を導出し概念の枠を評価する，FCA-MERGE の中心アルゴリズムの適用
3. 概念の枠に基づいた最終的な融合 Ontology の生成

図 5.3 は Ontology を融合するための理論全体の概観を表している．この理論では 2 つの Ontology と自然言語文書の集合 D を入力データとして得る．この文書は 2 つの Ontology に関連があり，そのため文書は Ontology に含まれる概念で記述される．その文書は，最終的に融合された Ontology で要求される目的のアプリケーションから得られるであろう．

小節 2.2 で記述したように，D に含まれる文書から概念記述 (*concept descriptions*) が導出される．これらの概念記述から，形式的な関係は導出される．文書からのこの情報の抽出は必要である．なぜなら双方の Ontology によってすでに分類された実体 (instance) が普通ないからである．しかし，この状況が与え

られれば，第一段階を飛び越えることができ，２つの形式的概念に対し入力として直接実体の分類を使うことができる．

我々の Ontology 融合手法の第二段階は，FCA-MERGE の中心的なアルゴリズムから構成される．その中心となるアルゴリズムでは，２つの関係を融合し，FCA 技術を使って融合された関係から概念の枠を算出する．

概念枠からの融合 Ontology の導出の最終段階は人間とのやり取りが求められる．よい結果を得るために，２，３の仮定が入力データに必要である．

1. 文書はソースとなる Ontology のそれぞれにとって適切である必要がある．各ソース Ontology に対して１つも事例を取り出せないような文書はこのタスクに対して無視される．この事実は，FCA-MERGE によって実行される応用駆動 (Application- driven) の方法を反映している．
2. 文書はソース Ontology から得られるすべての概念を網羅する必要がある．網羅されていない概念は，融合手続きの後に手動で扱われなければならない (あるいは文書の集合を拡張しなければならない)．
3. 文書は概念を十分にうまく分離できなければならない．もし異なると考えられている２つの概念がいつも同じ文書に現れるなら，FCA-MERGE は目標 Ontology において同じ概念へと写像する (ただし，この決定が knowledge engineer によって書き換えられない限り)．このような状況が頻繁に起こる場合，knowledge engineer はその概念をさらに分離するような文書をもっと付け加えることになるだろう．

続いて，入力 Ontology の集合に基づいて，FCA-MERGE を用いて一般的な Ontology を導出するための３つの段階を紹介する．

1.2.3 Extraction of Extensional Concept Descriptions

はじめに，各 Ontology $\mathcal{O}_i, i \in \{1,2\}$ に対し形式的関係 $\mathtt{K}_i := (G_i, M_i, I_i)$ を生成する．文書 D の集合は目標集合 $G_i := D$ として与えられ，概念の集合は属性の集合 $M_i := \mathcal{C}_i$ として与えられる．２値関係 I_i については，文書 g が m の実体 (instance) を含むときはいつも関係 $(g,m) \in I_i$ が保持されるようにする．

最後に，$\mathcal{H}^{\mathcal{C}}$ 関係に対する再帰性と推移性が formal concept にまとめられる．すなわち，$(g,m) \in I$ かつ $\mathcal{H}^{C}(m,n)$ は，$(g,m) \in I$ を意味している．

図 5.4 は，tourism domain から得られた２つの小さな Ontology の例に対し１４の文書から生成された関係 $\mathtt{K}_1$ と $\mathtt{K}_2$ を表している．

1.2.4 FCA-MERGE Core Algorithm

第２段階では，前段階で生成された２つの形式的な関係 $\mathtt{K}_1, \mathtt{K}_2$ を入力として得て，枝刈りされた (pruned) 概念の枠を返す．この枠は次のステップの入力として使われる．

始めに，２つの形式的な関係が２つの新しい関係K(これから枝刈りされた概念の枠が導出される) へと融合される．融合の前に，C_1 と C_2 は同じ概念を含んでいるので，その属性集合の曖昧さをなくしておく必要がある．つまり，$i \in \{1,2\}$ に対し $\tilde{M}_i := \{(m,i) \| m \in Mi\}$ とする．概念を指数化すると，両方の Ontology に同じ概念が存在するのに異なるように扱っている可能性が出てくる．たとえば，*CAMPGROUND* は１つ目の Ontology では *ACCOMMODATION* として考えられているのに２つ目の Ontology では違うかもしれない．したがって，融合された formal concept は $\mathtt{K} := (G, M, I)$ から得られる．ここで，$G := D, M := \tilde{M}_1 \cup \tilde{M}_2, (g,(m,i)) \in I :\Longleftrightarrow\ (g,m) \in I_i$.

K の全体の概念の枠は評価しない．これは，あまりに多くの特別な概念を提供すると考えられるからである．評価は，ソース Ontology の概念によって生成される少なくとも１つの formal concept の上にある formal concept に制限する．ソース Ontology の明確化の範囲内に残るものは保証する．さらに正確に，枝

刈りされた概念枠は，$\underline{\mathcal{B}}_p(\mathtt{K}) := \{(A,B) \in \underline{\mathcal{B}}(\mathtt{K}) \| \exists m \in M : (\{m\}', \{m\}'') \leq (A,B)\}$ によって与えられる．例として，枝刈りされた概念枠が図 5.5 に示されている．これは，６つの formal concept から成っている．全体の概念枠のうち，２つの formal concept が枝刈りされている．２つのソース Ontology と比較して，それらがあまりにも特化されているためである．

1.2.5 Lattice Exploration

これまでの段階 (事例抽出，関係の導出，関係の融合) はすべて自動的にできる一方，概念枠から融合された Ontology を導出するには，人間とのやりとり (interaction) が必要である．これは，domain expert の背景知識に強く依存するためである．

前段階の結果は，枝刈りされた概念枠であり，この結果から目標とする Ontology が導出される．枝刈りされた概念枠の各 formal concept は，目標 Ontology における概念あるいは関連の候補である．

文書は目標概念の生成には必要ない．よって，注意は formal concept の目標に制限される．この formal concept は，ソース Ontology の概念集合である．枝刈りされた概念枠の各 formal concept に対し，関連するキー集合が分析される．各 formal concept に対し，次のケースに分類できる．

1. 濃度１のキー集合を必ず１つ持つ
2. 濃度１のキー集合を２つ以上持つ
3. 濃度が０か１のキー集合を１つも持たない
4. 持っているキー集合が空集合である

目標 Ontology の生成は，始めの２つの状況のうち，片方に含まれるすべての概念から始まる．はじめのケースは最も簡単である．formal concept はソース Ontology の１つから，確実に１つの Ontology 概念によって生成され，knowledgr engineer とのやりとりなしに標 Ontology に含まれる．例としては，”VACATION_ 1”と”EVENT_ 1” とラベル付けされた２つの formal concept がある．

２つめのケースは，ソース Ontology の２つ以上の概念が同じ formal concept を生成する場合である．これは，その２つ以上の概念が目標 Ontology の１つの概念へと融合されることを示している．ユーザはどの名前を残しておくかを尋ねられる．たとえば，キー集合 { CONCERT_ 1} と { MUSICAL_ 2} は同じ formal concept を生成し，それゆえ融合される． キー集合 { HOTEL_ 1} ，{ HOTEL_ 2} そして { ACCOMMODATION_ 2} もまた，同じ formal concept を生成する．後者のケースは，同一の Ontology に２つの概念を持っているため興味深い．これは，文書の集合がこれら２つの概念を分離するのに十分な詳しさを与えていないということである．

knowledge engineer はその概念を融合するか目標 Ontology に異なるものとしてその概念を付け加える．区別されるべきだと考えられる融合概念が多すぎるならば，それは文書集合が十分な量ではないということである．ここまでの２つのケースにおけるすべての formal concept が処理されればソース Ontology によるすべての概念が目標 Ontology に含まれる．ここで，二つのソース Ontology によるすべての関係は目標 Ontology へとコピーされる．衝突や二重になっているものは Ontology engineer によって解決する．

次に，３番目のケースを考える．これに該当する formal concept は少なくとも２つの概念によってソース Ontology から生成され，目標 Ontology におけるあたらしい概念や関係の候補となる．目標 Ontology に概念や関係を追加するかどうかの決定はユーザにまかされる．キー集合 (formal concept の最小限の記述) は新しい概念の名前，もしくは新しい関係でもってつなげられるべき概念に対する示唆を提供する．その最小のキー集合が新しい概念と新しい関係についての最小の arity に対する最も短い名前を提供するので，最小の候補をもったキー集合だけが考慮される．

４番目の場合においては，一つの formal concept が確かに存在する (空集合はつねに一つのキー集合であるため)．この formal concept は目標 Ontology において新しい最も大きい概念，Root 概念を生成する．

この概念を受け入れるか拒否するかは knowledge engineer 次第である．ほとんどの Ontology ツールは最も大きい概念のようなものを必要とする．たとえば，図 5.5 において ROOT_1 と ROOT_2 とラベル付けされた formal concept がこれにあたる．

1.2.6 views on the Pruned Concept Lattice

概念枠の枝刈りの際，どこを重要な部分とみなすかについて数多くの視点がある．この小節ではまず前節でのケース 2 (異なる Ontology 概念が同じ formal concept を生成する) の取り扱いを補助する視点を 2 つ紹介する．

1. 異なる Ontology から取り出される concept を融合
   例) CONCERT_1 と MUSICAL_2 のペア
2. 単一の Ontology から取り出される concept を融合
   例) HOTEL_1 と ACCOMMODATION_1 のペア ($\mathcal{O}_1$ に関して)

2 Collecting, Importing & Processing Documents

前節では，我々の Ontology 学習の枠の中で異なるソートをされた Ontology を再利用し，取り込み，融合する仕組みについて述べた．本節では，web 文書の収集 (collecting)，取り込み，処理 (processing)，そして変換に焦点をあてる．以下，文書に対する次の 4 つの処理技術について詳しく述べる．

1. **ontology-focused crawler** を用いた web からの関係する文書の収集
2. **自然言語処理**技術を用いた文書の shallow processing
3. 不完全構造文書を標準化された関係表現へ変換するための **document wrapper** の使用
4. 言語的に，そして部分的に意味的に注釈付けされた文書の，Ontology 学習アルゴリズムの **relational representation** (関係表現) への変換

2.1 Ontology-focused Document Crawling

web から dmain-specific な Ontology を抽出し整備する仕事は，拡張されかつ適応されるべき core ontology が与えられるところから始まる．一般的に crawler は Web page を回収するプログラムであり，普通，サーチエンジンや Web cache によって使われる．

アルゴリズム crawler は，スタート文書の集合 A に含まれる各文書をダウンロードする．各文書は FCA-MERGE において使われたものと同じ抽出機構を用いて解析される．各文書に対する抽出機構の出力に基づいて relevancy measure $r(d)$ を算出する．現在の impremention においてはこの relevancy measure は，文書の中で参照されている concept の総数に等しく，次のように定義される．

定義 5.2(document relevance $r(d)$)
$L_d := \{L \in \mathcal{L} | \mathcal{L} \in \lceil\}$ かつ $C_d := \{C \in \mathcal{C} | \exists \mathcal{L} \in \mathcal{L}_\lceil : (\mathcal{L}, \mathcal{C}) \in \mathcal{F}\}$ とする．ある文書 $d \in D$ に対する document relevance の値は次式によって与えられる．

$$r(d) = |C_d|$$

もし relevancy $r(d)$ がユーザの定義した閾値 r_{min} を上回るなら，その明確化された文書が学習コーパス $\mathcal{D}$ に追加される．ある文書 d からスタートするすべての hyperlink は再帰的に探索される．

2.2 Shallow Text Processing using SMES

Ontology 学習は，Ontology 構造 $\mathcal{O}$ の抽出に焦点がおかれている．自然言語文書や辞書から正規性を抽出するために，文書は正規化された表現構造へと変換される必要がある．従って，正規性を抽出するための構造が必要である．ここで，Parser は単語のトークンや concept 間の関係を確立する．可能な範囲での単語の連なり方は数多く存在するので，関係の選択を限定したもとで parser は制限をもたなければならない．

ここでは，Ontology 学習の枠組みを，ドイツ語テキストに対する Shallow Text Processing に対する一般的な構造，名づけて SMES システムに頼っている．

図 5.6 は自然言語処理コンポーネントの構造全体を図示している．図 5.6 に見られるように，NLP の枠組みの構造は次の４つの主コンポーネントに分解される．(1) 大きい lexical database と有限状態文法からなる linguistic knowledge pool，(2) Ontology へのアクセスと，定義式 2.1 に従った，関連のあるドメイン固有の lexicon を持つ conceptual knowledge モジュール，(3) lexical level と clause level での構文解析に対して異なるモデルを構成する shallow text 処理エンジン，そして (4) いわゆる text-chart，結果の誘導と保存に対する一般的データ構造．

以降の小節においては構文解析技術の核や linguistic knowledge pool などの概観が示されている．

2.2.1 Core Technology

2.2.2 Linguistic Knowledge Pool

Linguistic Knowledge Pool は，７０万以上の単語の完全形 (entity lexica) と合成＆タグ付けされたルールを含んでいる．

2.2.3 STP on the Lexical Level

lexical level における Shallow Text Processing は次の４つのモジュールに分けられる．(i)Tokenizer, (ii)Morphological Analysis, (iii)Compound Analysis, そして (iv)part-of-speech Filter．この小節の残りの部分では，各モジュールについて短い定義が示されている．

2.2.4 STP on the Clause Level

clause level processing は Named Entity Finder, Clause Level Processing などの３つのコンポーネントに分けられる．

2.2.5 Heuristic Processing

2.3 Semi-Structured Document Wrapper

辞書はしっかりと注意深く手で作られたスソースであるので，純粋なテキストからの Ontology 学習を補助する core ontology としてよい出発点を与える．ここで辞書とは単語がアルファベット順に並んでいてその単語の意味やなどが載っている reference book のことを言う．実世界では HTML 文書上の free text(ただし固有の区切り文字で分けられたもの) から XML まで，辞書には様々な種類の format が現れる．したがって，この不均質な辞書表現を Ontology 学習の枠組みに取り入れるために，いわゆる wrapper が必要である．web context 上での wrapper の仕事は，HTML 文書などとして暗に格納されていた情報を，formal なデータ構造として明示的に格納された情報へと変換することである．wrapper の構築は手動で，もしくは半自動的な手法，完全に自動的な手法を使うことによって行われる．

ここでは，RDF-Schema を用いて辞書構造を再定義して使用している．この構造は，一般的な辞書表現で記述されるようなクラス (たとえば termEntry) やプロパティ (has DifinitionText など) の集合を含んでいる．図 5.11 は RDF に基づいた表現になるよう正規化された辞書項目の例である．この正規化表現は自然言語処理モジュールの入力として与えられる．辞書から $\mathcal{L}^C$, $\mathcal{C}$, $\mathcal{H}^C$ のような Ontology 要素を抽出するための仕組みは６章にかかれている．

2.4 Transforming Data into Relational Structures

Ontology 学習アルゴリズムはそれぞれが固有の入力形態を必要とするため，この小節で言語的に注釈付けされたデータの，そのアルゴリズムで処理できる format への変換を formal に定義する．伝統的な relational database モデルは次の定義を土台として使っている．

定義 5.3
X を有限で空でない**属性**集合とする．

- ▷ 各 $A \in X$ に対し，A のドメインと呼ばれる，空でない $dom(A)$ 集合が指定される．
  $dom(X) = \cup_{A \in X} dom(A)$.
- ▷ X 上の**組** (tuple) は関数 $\mu : X \rightarrow dom(X)$ とする．ここですべての $A \in X$ にたいして $\mu(A) \in dom(A)$.
  また，$Tup(X)$ は X 上のすべての組の集合である．
- ▷ X 上の**関係**は有限集合 $r \subseteq Tup(X)$ である．

この定義に基づいて，言語的に前処理された文書から Ontology 学習アルゴリズムの入力に関連する specific な関係への変換が定義される．一般的に，Ontology 的な背景知識がまったく利用できないときは語彙レベル (lexical level) で，ある種の背景知識が利用できるときは概念的レベル (conceptual level) で Ontology 学習アルゴリズムを適用する．以下では，両方のレベルにおける relevant input relations を提供する．

2.4.1 Relations on the Lexical Level

Ontology 学習の枠組みにおいて利用できる 3 つの表現を示す．
Lexical entry-lexical entry relation $r_{\mathcal{U}}$　concept $\mathcal{C}$ や lexical entry から concept への対応する写像 $\mathcal{F}$ の集合を利用せずに lexical level に制限される relation．lexical entry-lexical entry relation $r_{\mathcal{U}}$ は与えられた lexical entry 集合それぞれの間の関連性を記述する (定義 5.4).

定義 5.4
lexical entry-lexical entry relation は関数 $f_{\mathcal{U}} : \mathcal{L}^C \times \mathcal{L}^{\ C} \mapsto \mathbf{R}_0^+$
表 5.2 に lexical entry-lexical entry relation $r_{\mathcal{U}}$ の例が示されている．$f_{\mathcal{U}}$ の返す値は異なる評価方法に基づいている．簡単な例では，2 つの lexical entry が言語的，あるいは発見的なパターンで共起する場合，それらの共起性頻度は上昇する．
Document-lexical entry relation $r_{\lceil\updownarrow}$　ベクトル空間モデルに応じた”document space”をもつという考えによって動機付けられた relation.

定義 5.5
document-lexical entry relation は関数 $f_{\lceil\updownarrow} : \mathcal{D} \times \mathcal{L}^C \mapsto \mathbf{N}_0$
関数 $f_{\lceil\updownarrow}$ は文書 $d \in \mathcal{D}$ において，ある lexical entry が発生する回数をカウントする．
Lexical entry transaction relation $r_{\updownarrow\sqcup}$　言語的，あるいは発見的な relation において発生した lexical entry を表現.

定義 5.6
lexical entry transaction relation は $r_{lt} \subseteq \mathcal{L}^C \times \mathcal{L}^C$ として与えられる．
lexical entry 間の conceptual relation を導出するために，この transaction view を主に使用する．

2.4.2 Relations on the Concept Level

初めに conceptual levelで紹介する relationは，すでに lexical levelで紹介されている．concept-concept relationは concept 空間，つまり concept 間の関連性を次のように表現する．

定義 5.7(concept-concept relation r_{cc})
concept-concept relationは関数 $f_{cc} : \mathcal{C} \times \mathcal{C} \mapsto \mathbf{R}_l^+$
relation r_{cc} の拡張として，relation r_{clc} を定義する．

定義 5.8(concept/lexical entry-concept relation r_{clc})
concept/lexical entry-concept relationは関数 $f_{clc} : (\mathcal{C} \cup \mathcal{L}') \times \mathcal{C} \mapsto \mathbf{R}_l^+$
r_{clc} の例が表 5.3 に与えられている．
lexical entryに基づいた文書空間が定義 5.5 で紹介された．文書空間は同じように，与えられた concept $\mathcal{C}$ の集合に基づいて定義される．

定義 5.9(document-concept relation r_{dc})
document-concept relationは関数 $f_{dl} : \mathcal{D} \times \mathcal{C} \mapsto \mathbf{N}$,
表 5.4 に r_{dc} の例が与えられている．文書 d_1 :=doc1.html, d_2 :=doc2.html, d_3 :=doc3.html において起こる concept の数が示されている．document-concept relation と FCA-MERGE の基礎となる formal context との間にある類似性に注目するかもしれない．document-concept relationから起こる (1)・起こらない (0) の 2 値の決定よって頻度を簡単に代用することによって formal context は導出される．
最後の relationは，concept transaction relationR_{ct} である．この relationは lexical levelにおいて既に説明されたが，concept の binary pair を表現しており，次のように定義される．

定義 5.10(concept transaction relation r_{ct})
concept transaction relationは 2 値関係 $r_{ct} \subseteq \mathcal{C} \times \mathcal{C}$ として与えられる．
この relation の基本的考えは，共起する concept 間の関連性が表現されていることである．concept transaction relationは concept 間の非分類的関係の抽出を手助けする仕組みに対して，入力データとして与えられる．この仕組みついては，６章の 1.3 節に述べられている．

3 Conclusion

５章のまとめ...

Ontology and the Semantic Web

山肩 洋子

2002/08/24-27

Chapter 6 : ONTOLOGY LEARNING ALGORITHMS

オントロジ学習アルゴリズムを紹介。4 章で述べられているオントロジの学習サイクル[1]おいて、アルゴリズムはオントロジの抽出 (extract) **と維持** (maintenance) **をサポートする。5 章では、本書の枠組に背景知識として既存のオントロジを導入し、用いる手法について紹介した。ここで紹介する全てのアルゴリズムは、基本オントロジとしての概念構造が与えられていなくても動作する。しかし、もしある種の概念構造が利用できるならば、これらのアルゴリズムは既存の背景知識 (例えば概念階層** (concept hierarchy) **のような、既存の概念構造) の上にさらなる概念構造を積み重ねることにより、より良い結果を産み出す。**

1 章では、オントロジ構造 O に従って、オントロジを抽出するためのメカニズムを紹介する。本章は以下の 3 つの部分にわかれている。

1. **概念を参照する lexical entry と、その関係** (relation)
2. **概念階層** (concept hierarchy) **の形での分類法的関係** (taxonomic relation)
3. **概念の対の間における非分類法的関係** (non-taxonomic relation)

入力データとして「自由自然言語テキスト (free natural language text)」 **と「構造化辞書」を扱い、それぞれについてアルゴリズムを紹介。**

本章で扱う技法は以下の二つに別けられる。

1. **統計的、もしくは機械学習指向のアルゴリズム**
2. **パターンマッチングに基づく技法**

本章では、専門用語抽出 (term extraction)、**統計的階層的クラスタリング** (statistical hierarchical clustring)、**相関ルール** (association rule) **のような、既存のアルゴリズムや技法を、**Semantic Web **におけるオントロジ学習のために用いる。**

始めに述べたように、維持 (maintenance) **はオントロジエンジニアにとって最も重要な問題である。2 章で、以下の二つのオントロジ維持を紹介する。**

知識構造の放棄 (unlarning knowledge) オントロジの刈り込み (ontology pruning) **と呼ばれ、当該のドメインに関係の無い概念を探索し、削除する役割を担う**

オントロジ拡張と改良 (ontology extension and refinement) オントロジのインクリメンタルな拡張 (incremental extraction) **を扱う。**

これらのアルゴリズムはオントロジ抽出アルゴリズムと多くの共通点をもっているため、既存のオントロジ構造を拡張するために、1 章で紹介するオントロジ抽出アルゴリズムも用いる。

1 Algorithms for Ontology Extraction

オントロジ抽出のためのアルゴリズムを紹介する。

[1] import, extract, prune, and refine

1.1 Lexical Entry Extraction

本節では、入力データを分析することによって、概念や関係を指し示す **lexical entry** を抽出するための、さまざまな手法を紹介する。

Lexical Entries and their relationship to words and terms

まず始めに、単語 (word) と専門用語 (term) の違いを明確にし、これらと lexical entry や概念 (concept) に対する関係をそれぞれ述べる。

結論として、word が何であるか、あるいは term が何であるかの明確な定義は無い。一般的に、term の重要な特徴は、日常言語の word よりはより専門的なものであるということである。従って、これ以降で、term と word の間をこれ以上区別することはせず、**lexical entry** を word と term の両方の一般記述 **(general description)** として用いることとする。

A Note on Extracting Concepts

次の問題は、concept を抽出することと lexical entry を抽出することの違いである。もし特定の lexical entry が、概念として、あるいは既存の概念へのマッピングをもつ lexical entry としてモデル化されるならば、問題は特定ドメインのモデリング問題と考えることができる。そこで、オントロジエンジニアに対し、lexical entry を潜在的な概念あるいは関係が表面化したものと提案し、本章のアプローチでは、概念指標抽出のための半自動法 **(semi-automatic ways for the extraction of concept indicators)** のみを提供する。

1.1.1 Frequency Measure

概念を指し示すような、適切な lexical entry を抽出するための簡単な手法は、(言語処理された) ひとまとまりの文書、すなわちコーパス D から、単に lexical entry の出現頻度を数えることである。このアプローチは、特定ドメインのテキストの集合に頻出する、lexical entry は、概念の層表を指し示しているという仮定に基づいている。情報検索に関する研究によれば、単に出現頻度を数えるよりは、term に重み付けする方法の方がより効果的である。標準的な情報検索では、次の方法によって term に重み付けする。

- **lexical entry frequency** $lef_{l,d}$ は、文書 $d \in \mathcal{D}$ に出現する lexical entryl の出現頻度
- **document frequency** df_l は、コーパス D において、l が出現する文書の数
- **corpus frequency** cf_l は、コーパス D 全体での l の出現の総回数

一般的に $df_l \leq cf_l$ であり、$\sum_d lef_{l,d} = cf_l$ が成り立つことに注意する。lexical entry の抽出は、情報検索の尺度である **tfidf(term frequency inverted document frequency)** に基づいて行う。tfidf は、以下のようにして算出する。

[DEFINITION 6.1]

$lef_{d,l}$ を lexical entryl の lexical entry frequency とする。また、df_l を lexical entryl の全体的 document frequency とする。ここで、lexical entryl の文書 d に関する $tfidf_{l,d}$ を次のようにして算出する。

$$tfidf_{l,d} = lef_{l,d} * \log(\frac{|\mathcal{D}|}{df_l}) \tag{6.1}$$

tfidf は文書中の lexical entry の頻度に対し、それがほぼ全ての文書に出現している場合、その重要度が下がるような因子によって、重み付けする。すなわち、非常に稀、あるいは非常に頻繁に出現する term は、バランスを保っているような term よりも下に評価される。最後に、標準的な停止語 (stopword)[2] のリストにおいてあらわれる

[2] 文書に含まれる意味的な内容を持たない前置詞や冠詞などの一般的に機能語と呼ばれる単語．このような単語は文書集合全体にわたり高い頻度で出現するため，あまり検索には役に立たない．このため，停止語のリストをあらかじめ作っておき，文書や検索質問を内部表現に変

lexical entry を除いて、コーパス D のある文書に含まれる全ての lexical entry のリストを生成する。lexical entriy l の tfidf の値は次のようにして算出する。

[DEFFINITION 6.2]

$$\text{tfidf}_l := \sum_{d \in \mathcal{D}} \text{tfidf}_{l,d}, \qquad \text{tfidf}_l \in \mathbb{R}$$

ユーザは、tfidf_l が越えなければならない閾値 $k \in \mathbb{R}^+$ を設定する。

1.1.2 Lexical Entry Extraction via Dictionary Exploration

辞書は、オントロジ学習に対する適切な入力データと考えられる。辞書に載っている定義は、lexical entry に与えるべき意味的特徴について、多大な情報を持っており、各 `termEntry` は、ある概念を指し示す可能性のある lexical entry と考えることができる。最後に、事前処理された lexical entry が、オントロジエンジニアに対し、概念の候補として提案される。これらの、抽出された基本の entry に対し、以下で述べるパターンマッチング技法が適応される。

1.2 Taxonomic Extraction

概念の分類的組織化 (taxonomic organization of concepts) は、オントロジにとって重要な基礎となる分野である。taxonomy の自動構築において、top-down、middle-out、bottom-up などいくつかの方法論が提案されている。本章は、与えられたデータから概念の taxonomy を引き出すための技法を二つ紹介する。

1. 階層的クラスタリング技法 (hierarchical clustering technique)。
2. 言語的に事前処理された辞書の定義を入力とし、分類的関係を抽出するためのパターンマッチングを適応。辞書のような半構造的なデータにおいて、非常に有効に動作。

1.2.1 Hierarchical Clustering

クラスタリングとは、オブジェクトを、似たメンバーで構成されたグループに体系づける処理と定義することができる。一般的に、クラスタリングには以下の二つの代表的な方法がある。

非階層的クラスタリング 各オブジェクトが正確に 1 つのグループに割り当てられる。
階層的クラスタリング 1 より大きなサイズのグループは、より小さなグループで構成される。

階層的クラスタリングの方が、クラスタリングの階層を生成するため、非階層的アルゴリズムに比べ、より多くの情報を包含し、詳細なデータ分析において効力を発揮する。しかし、非階層的クラスタリングに比べ、時間と場所をとる。

Baseline Hierarchical Clustering

階層的クラスタリングの木は、各オブジェクトから始めて最も似たものをまとめていくことにより、bottom-up 的に作ることもできるし、あるいは、全てのオブジェクトからなる 1 つのグループから始めて、別のグループに分けるといった、top-down 的な方法によっても作ることができる。クラスタリングの最も重要な問題は、適切な計算手法と類似尺度を選択することである。

換する際に取り除いておく.

Computation strategies used in hierarchical clustering

ここでは、統計的階層的クラスタリングにおいてよいパフォーマンスを示す、3 つの関数 (単結合法 (最短距離法)、完全結合法 (最長距離法)、グループ平均について述べる。

単結合法 (single linkage means) 各クラスタ中のもっとも近いオブジェクト同士の距離。それぞれのクラスタ中からオブジェクトを 1 つずつ選ぶ全ての組合わせを調べ、その最も類似度の大きい対を選ばなければならない。単結合クラスタリングは、

完全結合法 (complete linkage) 二つのクラスタの間の類似度は、二つのクラスタのオブジェクトのうち最も似ていないメンバーの類似度により算出される。このように、二つのクラスタの類似度は、二つの最も似ていないメンバーの類似度ということになる。

グループ平均 (group-average) グループ平均は両者の中間と考えることが出来る

Similarity Measure

クラスタリングを行うためには、オブジェクト間の類似度を測る何らかの尺度が必要である。本稿にとって最も重要な二つの尺度は、cosine measure(Definition 6.3) と kullback leibler divergence(Definition 6.4) である。[3]

[DEFINITION 6.3]

二つのベクトル $\mathbf{x}$ と $\mathbf{y}$ の間の cosin measure または正規化相関係数は、以下のようにして算出される。

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\sum_{x \in X, y \in Y} xy}{\sqrt{\sum_{x \in X} x^2} \sqrt{\sum_{y \in Y} y^2}} \tag{6.3}$$

[DEFINITION 6.4]

確率度数関数 (probability mass fanction) $p(x), q(x)$ について、それらの関係エントロピーは以下のようにして算出される。

$$D(p||q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} \tag{6.4}$$

kullback leibler divergence は、二つの確率分布がどの程度独立であるかを測る尺度である。p と q との間の kullback leibler divergence は、完全に正しくない分布である q に基づいて作られたコードにより、分布 p をエンコードしたとき失われるビット数の平均である。その値は常に正であり、$p = q$ のときのみ $D(p||q) = 0$ となる。重要なのは、kullback leibler divergence が $p(X) > 0$ 且つ $q(x) = 0$ の時定義されないという点である。オブジェクトの確率分布が多くの 0 値を含む場合、bottom-up クラスタリングはほぼ使うことが出来なくなる。よって、lullback leibler divergence を使う場合は、top-down クラスタリングを行う方がより自然である。

[Example] 次のような concept-concept matrix を考える。

cosin measure 概念 HOTEL のベクトルを $\mathbf{x}^T = (0, 14, 7, 4, 6)$ とし、概念 ACCOMODATION のベクトルを $\mathbf{y}^T = (14, 0, 11, 2, 5)$ とすると、

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{7 \cdot 11 + 4 \cdot 2 + 6 \cdot 5}{101 \cdot 150} \approx 0.93 \tag{6.5}$$

[3] これ以外の類似尺度は (Manning and Schetze, 1999) を参照のこと。

表 1: 図 6.1 行列 r_{cc} の例

ID	Hotel	Accomodation	Address	Weekend	Tennis
Hotel	-	14	7	4	6
Accomodation	14	-	11	2	5
Address	7	11	-	10	3
Weekend	4	2	10	-	5
Tennis	6	5	3	5	-

kullback leibler diversion まず始めに、各概念について確率度数関数を算出する。概念 HOTEL に関する確率度数関数は、$(0, 0.45, 0.22, 0.13, 0.19)$ となり、概念 ACCOMODATION に関する確率度数関数は、$(0.44, 0, 0.34, 0.06, 0.16)$ となる。よって、kullback leibler diversion は

$$D(\text{Hotel}||\text{Accomodation}) = 0.22 \cdot \frac{0.22}{0.34} + \cdots + 0.19 \cdot \frac{0.19}{0.16} \approx 0.65 \tag{6.6}$$

Hierarchical clustering including background knowledge

階層的クラスタリングアプローチの問題は、計算されたクラスタは、ラベル付けされていない、つまり、クラスタは複数概念の結合として表現され、その後、オントロジ技術に基づいてラベル付けされなければならないということである。よって、既存の分類法的背景知識に基づいて、算出されたクラスタにラベル付けする。

1.2.2 Pattern-Based Dictionary Exploration

言語的に処理された辞書の定義に対し、パターンマッチングのアプローチを適応する。基本的なアイディアは、正規表現を用いて繰り返し発生する表現を取りだし、その結果を $\mathcal{H}^{\mathcal{C}}(C_1, C_2)$ のようにして、意味構造に位置づけることである。

このような、辞書を用いたオントロジ学習では、自然言語成分が正規的に出現するという理由から、パターンが非常にうまく動作する。このアプローチのアイディアは、特定ドメインの辞書に含まれた、構造化された情報を、taxonomic 関係を抽出するための入力として用いることである。

1.3 Non-Taxonomic Relation Extraction

1.3.1 Core Learning Algorithm

core learning algorithm は、与えられたデータベースから相関規則 **(association rule)** を抽出するというアイディアに基づいている。相関規則はデータマイニングの分野で確立され、膨大なデータ項目のなかから相関規則を見つけてくる手法である。典型的な例として、market basket analysis がある。客がショッピングバスケットの中にいれた様々な物の間から、相関を見つけることにより、客の買い物習慣を分析する。相関規則から発見された情報は、スーパーマーケットの配置の最適化し、マーケティングストラテジーを確立する手がかりとなる[4]

このようなアイディアが、non-taxonomic relation を抽出するためのオントロジー学習とどのような関係があるのだろうか?基本的なアイディアは、Definition 5.10 で定義したような、言語的出力の確率的情報を、関連規則を用いて分析することである。たとえば、言語的な処理を行った結果、`"Costs at the youth hostel amount to $20 per night"` というように、`"cost"` という lexical entry と同じ文に、`"hotel"`,`"guest house"`,`"youth hostel"` と

[4] ミルクとパンを近くに配置すると、一度の買い物で両方を同時に買ってくれる可能性が高まる、など。有名な例としては、客の買い物を分析した結果、おむつとビールが同時に購入される場合が多いことから、両者を近くに写したところ、量売り上げが大幅に延びた

いう lexical entry が発生していることがわかったとする。すなわち、HOTEL,GUETS HOUSE, YOUTH HOSTEL という概念と、COSTS との対応が再発するという統計的なデータが示されたことになる。この例では、HOTEL と COSTS の関係をオントロジーに含むようエンジニアに提案する。

Input for Non-Taxonomic Relation Extraction]

言語レベルでは、Definition 5.6 に従い、lexical entry transaction relation が用いられる。概念レベルでは、concept transaction relation が、メカニズムに対する入力として用いられる。これらのペアは、以下で述べる相関関係アルゴリズムの入力として用いられる。

基本的な相関関係アルゴリズムでは、transaction の集合 $T := \{t_i | i = 1 \cdots n\}$ が与えられる。ここで、各 transaction t_i は、アイテム (item) の集合である。もし、アルゴリズムが言語レベルで実行されれば、このアイテムは $t_i := \{a = i, j | j = 1, \cdots m_i, a_{i,j} \in \mathcal{L}^{\mathcal{C}}\}$ として与えられ、各アイテム $a_{i,j}$ は lexical entry $\mathcal{L}^{\mathcal{C}}$ の集合である。一方、アルゴリズムを概念レベルで実行すれば、アイテムは $t_i := \{a_{i,j} | j = 1 \cdots m_i, a_{i,j} \in \mathcal{C}\}$ として与えられ、各アイテム $a_{i,j}$ は概念 C の形で得られる。

アルゴリズムにより、指示度 (support) と確信度 (confidence) がユーザが定義した閾値を越えるような、lexical entry の相関関係 $X_k \rightarrow Y_k (X_k, Y_k \subset \mathcal{L}^{\mathcal{C}}, X_k \cap Y_k = \emptyset)$、もしくは概念の相関関係 $X_k \rightarrow Y_k (X_k, Y_k \subset \mathcal{C}, X_k \cap Y_k = \emptyset)$ を算出する。ここで、ルール $X_k \rightarrow Y_k$ の指示度は、$X_k \cup Y_k$ を部分集合として含むような transaction のパーセンテージであり、$X_k \rightarrow Y_k$ の確信度は、X_k が transaction に現れたときに Y_k が同時に現れる transaction のパーセンテージで定義する。

[DEFINITION 6.5] (Support of an association rule)

$$\text{support}(X_k \rightarrow Y_k) = \frac{|\{t_i | X_k \cup Y_k \subseteq t_i\}|}{|T|} \tag{6.7}$$

[DEFINITION 6.6] (Confidence of an association rule)

$$\text{confidence}(X_k \rightarrow Y_k) = \frac{|\{t_i | X_k \cup Y_k \subseteq t_i\}|}{|\{t_i | X_k \subseteq t_i\}|} \tag{6.8}$$

相関関係を算出する一般的な方法は、以下の 2 ステップに分けられる。

1. 指示度が、ユーザが定義した最小指示度 s_{min} 以上であるような、全てのアイテムセットを算出する。これらのアイテムセットを frequet item set と呼ぶ。
2. これらの frequent item set から、確信度がユーザの定義した最小確信度 c_{min} を越えるものを選ぶ

事前処理されたデータから frequent item set を抽出するための基本的な方法が、Algorithm 3 に示されている。

アイテムセット $X' \subseteq X$ について、$\text{support}(X') \leq \text{support}(X)$ が保たれているため、frequent item set を $l_1, l_2, \cdots$ の順で順番に計算してゆくことにより、効率よく動作するアルゴリズムである。

最後に、以下のような特性を用いて confidence を算出する。X を frequent item set とすると、全ての $X' \subseteq X$ について

$$\text{confidence}((X - X') \rightarrow X') = \frac{\text{support}(X)}{\text{support}(X - X')} \tag{6.9}$$

1.3.2 Relation Extraction including Background Knowledge

taxonomy から得た背景知識利用することにより、適切なレベルの抽象度で、関係を提案することができる。概念間の一般化関連規則を生成するのに、背景知識を用いるメリットを、以下での例で簡単に説明する。

言語的処理を行った結果、"hotel","guest house","youth hostel" という lexical entry が、"costs" という lexical entry と一緒に用いられていた、つまり、"Costs at the youth hostel amount to $20 per night" というような文で現れていたことがわかった。すなわち、統計的データによって、概念 HOTEL,GESTHOUSE, YOUTHHOSTEL が、COSTS と同時に発生することが示せる。さらに、taxonomic relation $\mathcal{H}^{\mathcal{C}}$(HOTEL, ACCOMODATION), $\mathcal{H}^{\mathcal{C}}$(GUESTHOUSE, ACCOMODATION), $\mathcal{H}^{\mathcal{C}}$(YOUTHHOSTEL, ACCOMODATION) が、これらの概念間に存在する。したがって、一般化相関規則アルゴリズムは、これらの 3 つのペアの間の関係について、確信度・指示度を算出することによって、同時に ACCOMODATION と COSTS のような、より抽象の高いレベルでの関係も得られる。最後に、適切でない関係を刈り取ることにより、概念の関係を記述するのに最も適した抽象レベルを決定する。この例だと、ACCOMODATION と COSTS との関係が、オントロジに加えるよう提案される。背景知識を利用することにより、sub-concept に継承されるような、より一般的な non-taxonomic relation を生成することができる。たとえば、ACCOMODATION と COSTS との間には、HASCOST という関係があるが、同時に ACCOMODATION の sub-concept と COSTS との間にもその関係がある。

non-taxonomic relation を、背景知識を用いて生成するアルゴリズムを Algorithm 4 に示す。まず始めに、各 transaction t_i をそれぞれ、概念 $a_{i,j}$ の祖先を含むよう、$t_i' \cup \{a_l | (a_{i,j}, a_l) \in \mathcal{H}^{\mathcal{C}}$ と拡張する。つぎに、この "pre-compiled" data について、すべての可能な関連規則 $X_k \rightarrow Y_k$ の確信度と支持度を算出する。ここで、Y_k は X_k の祖先を含まない[5]。最後に、"ancestral" rule $\hat{X}_k \rightarrow \hat{Y}_k$ に包含されるすべての関係規則 $X_k \rightarrow Y_k$ を刈り取る。ここで、item set $\hat{X}_k, \hat{Y}_k$ は、$X_k \rightarrow Y_k$ において、item set X_k, Y_k の祖先もしくは同一の item のみを含む。

【例】

WWW から、ツーリストインフォメーションのテキストコーパスをとり込み、事前処理する。このコーパスには「場所、宿泊、家具、管理情報、伝統行事」を参照するオブジェクトが記述されている。

例文 (7a) と (7b) から lexical entry 間の依存関係が抽出された。文章 (7c) と (7d) からは、前置詞句による接続のヒューリスティックと、文のヒューリスティックが lexical entry の対をそれぞれ関連づけている。このように、用語集から得た知識により、4 つの概念ペアが導かれた。

一般化関連規則を学習するアルゴリズムでは、図 6.4 のような概念階層と、上で述べらたような概念ペアを用いる。アルゴリズムを実行すると、interesting かつ重要な non-taxonomic 概念関係がかなりの数見つかる。それらのごく一部を表 6.3 に示す。

ここで、(AREA,HOTEL) や (ROOM,TELEFVISION) といった二つの概念ペアが、ユーザに提示されることなく、刈り取られていることに注意する。その理由は、その祖先の関係規則である (AREA,ACCOMODATION) や (ROOM,FURNISHING) が、それぞれ高い確信度と指示度を得ているからである。

1.3.3 Concept Constraints $\mathcal{C}^*$

1.3.4 Hierarchical Ordering of Extracted Relation

概念の taxonomy の形の背景知識を用いると、冗長で不必要な多くの関係を生成することになる。次のような例を考えてみよう。まず、入力データから関係 (Hotel, Room) が動機づけられ、アルゴリズムにより一般化関係 (ACCOMODATION,ROOM) が生成されたと仮定する。ユーザはこれらの関連をオントロジにどのようにして加えるか決定しなければならない。1 つの解決策は、HASROOM(ACCOMODATION,ROOM) と、HASHOTELROOM(HOTEL,ROOM) という関係を導入することである。さらに、オントロジエンジニアは HASROOM と HASHOTELROOM の間の関係を、自明の理を用いて定義する。つまり、HASHOTELROOM は

[5]妥当なのは明白なので。

HASROOM の sub-relation であると言える。この決定課程をサポートするため、関係の順序を定義しなければならない。taxonomy$\mathcal{H}^C$ によって得られた順序により、抽出した関係を階層的に表現する。

[DEFFINITION 6.7] (Predecessor Relation)
(C_1, C_2) を、抽出された関係とする。$(C_1'' \in \text{Clos}(C_1, \mathcal{H}^C) \wedge (C_2' \in \text{Clos}(C_2, \mathcal{H}^C)$ であり、且つその場合のみ、(C_1', C_2') は (C_1, C_2) の predecessor relation である。
図 6.5 の例では、(HOTEL,DOUBLEROOM) の関係は、(ACCOMODATION,$\mathcal{H}^C$) の predecessor relation である。なぜならば、HOTEL $\in$ Clos(ACCOMODATION, $\mathcal{H}^C$) であり、かつ DOUBLEROOM $\in$ Clos(ROOM, $\mathcal{H}^C$) であるからである。

[DEFFINITION 6.8] (Direct Predecessor Relation)
もし、$(C"_1, C"_2)$ が (C_1, C_2) の predecessor relation であり、かつ (C_1', C_2') が $(C"_1, C"_2)$ の predecessor relation であるような $(C"_1, C"_2)$ が存在しなければ、(C_1', C_2') を (C_1, C_2) の direct predecessor relation である。

上記の例を考えると、(HOTEL, ROOM) は (ACCOMODATION, ROOM) の direct predecessor relation である。定義 6.7 や 6.8 で与えられた predecessor、あるいは direct predecessor relation は、抽出された関係の階層的な表現を生成するのに用いられる。taxonomy における各関係を階層的な表現にするためには、有り得る全ての predecessor relation を算出し、それを"superrelation"として関連づける。実際には、direce predecessor relation だけ算出すれば十分である。
図 6.5 は、抽出された関係 (ACCOMODATION,ROOM), (HOTEL, ROOM),(ACCOMODATION,DOUBLEROOM), (HOTEL,DOUBLEROOM) の階層的順序と、オントロジーエンジニアにより加えられたオントロジ要素が示されている。

2 Algorithms for Ontology Maintenance

本章では、オントロジ学習サイクルの中でも、データ指向のオントロジメンテナンス手法を紹介する。これは、オントロジを特定のアプリケーションに適応しあつらえ、アプリケーションのデータを分析することにより、これを拡張するフェーズである。

2.1 Ontology Pruning

2.1.1 Baselin Pruning

概念、あるいは関係を語彙化したものが、頻出していない、あるいは出現さえもしないならば、その概念や関係はオントロジから省略してよいと言える。まず、ドメインのテキストを処理し、各概念ごとに (テキストに含まれる概念の具体的語彙を、概念にマッピングすることにより) ドメイン頻度が算出される。さらに、頻度が上位概念に伝搬される。もし概念が特定ドメインコーパスに頻出するならば、その概念と、その上位概念はオントロジに残す。

2.1.2 Relative Pruning

The Relative Pruning Algorithm
基本的なアイディアは、与えられたオントロジについて、一般化コーパスから得られた関連頻度と、ドメインに

関する関連頻度と比べることである。このプルーニングアルゴリズムを Algorithm 5 に示す。ここでは、オントロジ O が入力とし、特定ドメインコーパス D、一般化コーパス G、比率 $r \in \mathbf{R}^+$ と、選択した頻度算出法 m とする。

まず全ての lexical entry について、ユーザが選択した尺度により頻度を算出する。次に、二つのコーパスから得られた頻度が比較される。特定ドメインコーパスに、より頻出するような全ての概念や関係はオントロジに残される。ユーザは同時に、特定ドメインにも一般ドメインにも含まれないような概念を、オントロジから刈り取るべきかどうかを特定することができる。

プルーニングを行っている間、lexicon $\mathcal{L}^{\mathcal{O}}$ におけるマッピング情報もまた更新しなければならない。一般的に、以下の二つの方法をとる。1 つ目は、ユーザが lexical entry に起こすことを決定しなければならないものである。2 つ目は、自動手法である。F の関係の損失を最小化するため、オントロジーに残っている $C \in \mathcal{C}$ の中の最も近い上位概念をマージする。複数のパスに上位概念が残った場合は、その幹となる関係を削除する。なぜならば、正しい上位概念を自動的に選択することができないからである。例えば、もし"chair"がオントロジから刈り取られてしまったら、"furniture"に対する lexical reference も、オントロジに残っている"chair"の最も近い概念へと更新される。

2.2 Ontology Refinement

本章では、新しい概念を取り込むことにより、オントロジを増加拡張する手法を紹介する。抽出のためのアルゴリズムは、オントロジの改善 (refinement) と拡張 (extension) のためにも用いられるが、抽出は、主に全体のオントロジーの協調的なモデリングに役立つが、改善はターゲットとしているオントロジをうまく調整し、その発展をサポートする。本章では、未知の lexical entry の意味を抽出し、かつ、ある lexical entry が新たな概念への参照として定義されるべきであるという指標を発見することを目的とする。このアプローチは、未知の lexical entry が、すでに概念とマッピングされているような既存の lexical entry と、似た概念的振る舞いを共有することを想定している。

Example. 未知の lexical entry "weekend excursion"を見ると、だれもが EXCURSION という概念となんらかの振る舞いを共有していると言うことができる。概念的な振る舞いを共有することは、1.2.1 節で導入した概念ベクトルに基づいて、分布の類似度尺度を用いることにより算出される。chapter 5 で導入された Definition 5.8 によると、この学習タスクのためのデータは、図 6.4 に示すような concept/lexical entry-concept matrix として表される。

ID	Accomodation	Hotel	Event
Excursion	5	4	2
"weekend excursion"	4	4	1
Car	1	1	1

2.2.1 The Refinement Algorithm

改善 (refinement) のためのアルゴリズムの基本的なアイディアは、1) 重要な未知の lexical entry が組織化され、2) 類似の概念が取り出されユーザに提示され、3) 適切に割り当てることにより、未知の lexical entry の意味をユーザが決断する。増加改善アルゴリズムを Algorithm 6 に示す。concept/lexical entry matrix、オントロジ、選択された類似尺度、二つの閾値が入力として与えられる。

r_{clc} の中の、未知の lexical entry $L'_i \in U$ (U は概念へのマッピングが未知の lexical entry の集合) の連結頻度が計算される。これは、概念に関する全体の発生回数を足し合わせることによって行われる。表 6.4 で与えられた例によれば、未知の単語"weekend excursion"の連結頻度 cf は、cf("weekendexcurtion") = 9 となる。もし連結

頻度が閾値 t を越えていれば、未知の lexical entry と、概念集合 C に対応する全ての既存の lexical entry との類似度を算出する。最も類似した k 個の概念が、オントロジエンジニアに提示され、オントロジエンジニアは、既存の概念へ割り当てる、あるいは新たな概念を定義し、未知の lexical entry をその概念にマッピングすることにより、未知の lexical entry の意味を決定する。

アルゴリズムを更に拡張するため、もし $\mathcal{H}^C$ に含まれる taxonomic の知識が利用できるならば、未知の lexical entry と既存の概念の全体的な結合頻度の判定を改良するのに用いることができる。例えば、lexical entry もしくは概念が、HOTEL と結び付けられるならば、HOTEL は、taxonomic relation $\mathcal{H}^C$(HOTEL, ACCOMODATION) により、自動的に概念 ACCOMODATION と結合される。

3 Conclusion

この chapter では、オントロジの抽出と維持を行うアルゴリズムを紹介した。アルゴリズムは、一方では (lexical entry 抽出、lexical entry の階層的順序、lexical entry 間の関連の抽出などによる) オントロジの発展をサポートし、もう一方では、(概念マッピングや、概念階層の lexical entry という形の) 既存の概念的背景知識から利益を得た。

以下では、本章では部分出来に語られただけであったが、今後重要になる問題について述べる。

3.1 Multi-Strategy Learning

multi-strategy learning は、supervised learning task において非常にうまく動作した。基本的な考え方は、まず始めに簡単なアルゴリズムで基本構造を探索し、徐々に複雑なアルゴリズムに適応することにより、その探索範囲を縮めるというものである。

3.2 Taxonomic vs. Non-Taxonomic Relations

研究はされているが、taxonomic と non-taxonomic の違い (どちらかがどちらかに継承されるか？など) についてさえ、まだわかっていない。

3.3 A Note on Learning Axioms – $\mathcal{A}^0$

axioms の定義とメンテナンスは、オントロジエンジニアにとって重要な基礎であるが、現在のシステムでは、ユーザは axioms を手動でモデル化するしかない。将来的には、axiom を半自動的に抽出するメカニズムが必要となるだろう。axiom を抽出するためのアルゴリズムは、良く構造化された relational data(最も良いのは knowledge base) が必要である。この種の relational data は現在の時点では稀だが、Semantic Web の成功と共に増加するだろう。

IMSS-2002
2002 年 8 月

TEXT-TO-ONTO 環境

西口敏司

7　TEXT-TO-ONTO 環境

本書における方法論的、理論的研究結果が総合的なオントロジ学習環境 **TEXT-TO-ONTO** に実装された。**TEXT-TO-ONTO** はまだ、途中結果である。**TEXT-TO-ONTO** は大きくわけて 2 つの部分に分けることができる。本書の PART I では、オントロジ、オントロジの設計、オントロジに基づく応用の基礎が導入された。これらに基づいて、オントロジの手動設計および管理環境である、**ONTOEDIT** が設計された。

PART II では、セマンティック Web のためのオントロジ学習を支援するための構造、コンポーネント及び具体的な機構が提供された。バランスの取れた協調モデル化のアイデアに触発され、**ONTOEDIT** はデータのインポートと処理技術、そしてオントロジ構造の抽出とメンテナンスを支援するアルゴリズムの点で拡張された。

ONTOEDIT のようなオントロジ設計環境において、オントロジ学習をプラグインとして利用することに対して、実世界のシナリオに用いるときには、データのインポート、処理、アルゴリズムの実行に対する機構とユーザインターフェースが特に重要である。

TEXT-TO-ONTO は総合的なオントロジ管理と設計環境上に成り立っており、半自動化オントロジ抽出と様々な種類の入力データの上の保守に対する手段を提供する。

本章では、オントロジ学習環境である **TEXT-TO-ONTO** について述べる。

本章は 3 つの節に分かれている。

第 1 節では、環境の実装の基礎となる、コンポーネントに基づくアプローチが導入される。3 つの異なる型のコンポーネント (データ構造コンポーネント、処理コンポーネント、インターフェイスとユーザインタラクションコンポーネント) に分かれる。

次に、前に述べたように、**TEXT-TO-ONTO** は手動でオントロジを設計するための環境である、**ONTOEDIT** がその基盤である。第 2 節では、**ONTOEDIT** の機能を説明する。**ONTOEDIT** は、第 3 章で紹介された、階層化されたオントロジ設計手法に従う。従って、どのように用語エントリ L がこの環境を用いて集められるかが説明される。続いて、これらの用語エントリに基づいて、どのように概念 C がモデル化され、分類学 HC において組織化されるかが示される。オントロジ設計における重要な点は、GUI を用いて手動で設計される、概念や公理間の非分類学的関係の定義の支援である。公理はオントロジ設計において重要な構成ブロックであると考えられる。公理の自動的な抽出および提案に対する手段はオントロジの設計者には提供されていないが、グラフィカルな手段を用いて、抽出された構造に基づいて公理を定義する機構は導入されている。さらに、与えられたオントロジに基づいて **ONTOEDIT** を用いて、どのようにして実体 I とそれらの間の関係が定義されるかを示している。また、どのようにして **ONTOEDIT** が **SiLRI**、つまり推論エンジンに基づく **F-Logic**、に接続するかも説明されている。

どのようにオントロジと、関連する知識ベースが検索されるかという例が与えられ、置換が **F-Logic** の意味に従って計算される。

全体的なアプローチの主な基礎をなすアイデアは、それぞれのモデリングステップが利用者またはオントロジ学習アルゴリズムによってなされる可能性があることである。

従って、第 3 節では **ONTOEDIT** 上に構築された **TEXT-TO-ONTO** の機能を説明する。この節は、**TEXT-TO-ONTO** の管理コンポーネントによってアクセスされるデータのインポートや処理コンポーネントの主な特徴を説明することから始める。

引き続き、利用者がどのようにしてライブラリに含まれるオントロジの抽出やメンテナンスに対するアルゴリズムにアクセスするか、いくつかの例が示される。重要な点は、アルゴリズムの抽出とメンテナンスの提案の適切な表現である。

結果の表現に対する利用可能かつ **TEXT-TO-ONTO** 上に実装された異なるビューが説明される。

最後に、本章をまとめる前に、オントロジ設計と学習環境を改良するための将来の仕事の重要な問題のリストが与えられる。

7.1 コンポーネントベースのアーキテクチャ

TEXT-TO-ONTO 環境の主なコンポーネントと、それらの間のインタラクションを、図 7.1 に示す。読者はオントロジ学習フレームワークとそれに伴う構造が、第 4 章で導入されたことを知っている。前に述べたように、構造の実装において、データ構造、処理モジュール、GUI は、内部的に分割されている。

すべてのコンポーネントは、環境全体において重要な役割りを演じる、オントロジ、知識ベース、用語モデルで結びつけられている。

これらの 3 つのモデルは、第 2 章で導入した形式的定義の実装である。3 つの異なるタイプのコンポーネントに分割される。

データ構造コンポーネント 用語、オントロジと知識ベースのモデルは、定義 2.1 で導入されたオントロジ O と、知識ベース構造 KB の実装である。

処理コンポーネント **TEXT-TO-ONTO** における処理コンポーネントは、自然言語処理環境、推論エンジン、異なるオントロジ表現言語を支援するいくつかの入出力コンポーネントである。

ユーザインタフェース及びインタラクションコンポーネント インタフェイスとユーザインタラクションコンポーネントは、**ONTOEDIT** と **TEXT-TO-ONTO** 操作 GUI である。これらは、ドキュメントブラウザ、パラメータ設定インタフェース、結果表示ブラウザを含む。

処理コンポーネントは、第 5 章と第 6 章で導入した技術とアルゴリズムの実装である。

ONTOEDIT を利用した手動によるオントロジの設計方法と、**TEXT-TO-ONTO** に含まれるオントロジ学習技術の用法の間には厳密な線を引くことができない。これは、バランスのとれた強調モデル化のパラダイムを反映している。

7.2 オントロジ設計環境 ONTOEDIT

ONTOEDIT は、第 3 章で導入したオントロジ設計方法の実装である。**ONTOEDIT** は、階層化オントロジ設計パラダイムに従ってオントロジの開発を支援する。

用語法 (The Lexicons) オントロジの設計は、概念、関係、実体を参照する用語の集合から始める。

図 7.3 に、オントロジ O と知識ベース KB に含まれるコア原始プリミティブ C, R, I の用語的表現の定義を支援する表示を示す。

用語エントリとコアプリミティブの間に m:n 対応が支援されていなければならない。

- 自然言語処理に関わるタグ (POS)

- GUI に関するタグ (それぞれのオントロジに基づくアプリケーションの GUI に示される用法エントリの特別な定義)

図 7.3 に示す用語エントリは既に自然言語処理環境 SMES とのインタラクションの観点から、それ自身の語幹が取り出されている。

概念 C、概念の階層 HC、関係 R 用語エントリ L の先頭において、概念と関係の集合が定義される。図 7.4 に概念的分類 HC におけるモデリング概念とその操作に関する表示を示す。

概念は多重継承を用いて定義される。多重継承の概念を検査するための特別なビューは「More..」ボタンを押すことによって表示される。このボタンを押すことによって新しいウィンドウが表示され、選択して与えられた概念のすべての上位概念と用語的エントリを表示する。

非分類学的関係の表示は、異なるビューを用いて定義される。図 7.5 に、**ONTOEDIT** で関係を定義するの際に利用可能な 2 つのビューを示す。図 7.5 の右側において、関係のドメインや範囲を定義することなく、一意識別子を持つ一次オブジェクトとして定義される。典型的に、関係の一意識別子は視覚化されず、その代りに特定の関係のための用語的エントリが利用される。

図 7.5 の中央の図に、特定のドメインの概念が割当てられた関係を示す。この図では、オントロジの設計者はドメインと範囲の制限を持つ新しい関係を定義したり、既存の関係にドメインや範囲の制限を追加することができる。

図 7.5 には、SWRC(Semantic Web Research Community) のために開発されたオントロジを引用した関係を示している。

ACADEMICSTAFF という概念が概念階層ビューに表示されている。灰色で示された関係は **ACADEMICSTAFF** の概念の上位概念から継承されている。

NAME という関係は、より一般的な概念 (例えば **PERSON** という概念) から構成される。継承される。関係の数は、とりわけ **ACADEMICSTAFF** という概念に割当てられている。例えば、例えば、**WORKSATPROJECT** という関係は、**PROJECT** という範囲を持つ。オントロジ O の概念的な定義とは反対に、**ONTOEDIT** は、**XML-Schema** 標準に含まれるデータ型を参照した、**STRING** や **INTEGER** のようないくつかの既定義のデータ型を提供する。

オントロジの設計者は、具体的な関係に対してメタな情報を定義することができる。例えば、最大、最小濃度、多言語文書、さらに多くの用語など。

ONTOEDIT を用いた公理の設計 オントロジ設計において重要な点は、意味的制約を保証し、かつデータの完全なビューを生成する公理の定義である。

前に述べたように、与えられた Web データからの公理の抽出を支援するオントロジ学習アルゴリズムは存在しない。しかし、意味的パターンを用いたアプローチに基づく公理の手動設計は考えることができる。

ONTOEDIT は、概念レベルの公理を定義するために、公理をモデル化し、かつオントロジ設計を支援する適切なユーザインタフェースを持ったグラフィカルな手段を提供する。

小さなパターンを具体化する 2 つの例が与えられる。すなわち、**逆関係 (inverse relations)** と、**互いに素な概念 (disjoint concepts)** である。

逆関係 (Inverse Relations) 逆関係の簡単な例は、第 3.9 節で次のように与えられている。

> 概念 **PERSON** と概念 **PROJECT** の間に、**WORKS_AT** という関係が存在している。そして、**PROJECT** と **PERSON** の間に **HAS_PARTICIPANT** という関係が存在している。互いに逆な 2 つの関係を明示的にモデル化することには、いくつかの利点がある。

- 知識ベースにおける一貫性を保証する
- 利用者を冗長な情報から解放する

図 7.6 に「関係公理」をモデル化する一般的なビューを示す。観光ドメインのための、いくつかの具体的な逆関係公理である。

左側には、ドメインや範囲制限に対応するすべての関係が列挙されている。右側には利用者に対して、2つの関係が逆であることを定義する表が示されている。利用者は左のペインから「+」ボタンを押すことによって、関係を追加することができる。逆関係は大域的に定義することができる。例えば、**IN_GEBIET** と **BIETET_UNTERKUNFT** は、一般に逆関係である。また、**IN_GEBIET(UNTERKUNFT, GEBIET)** と **BIETET_UNTERKUNFT(GEBIET,UNTERKUNF** は互いに逆関係であるというように、局所的に制限される。

大域的、局所的逆関係は、それぞれ、個々のドメインを制限する選択によって区別される。

互いに素な概念 図 7.7 に、いわゆる**概念公理**をモデル化するための一般的なビューを示す。

「互いに素な公理 (disjoint axioms)」はオントロジと知識ベースの一貫性と品質を強化する、特定の概念公理である。オントロジの設計者は、互いに素な概念をモデル化するためのビューを用いて、左のペインから概念を選択して、互いに素な性質を明示的にモデル化する。

もし、2個より多くの概念が選択されたとき、選択された概念の間の素な性質のデカルト積が計算され、特定の互いに素な概念が図 7.7 の右のペインに表示される。

オントロジの設計者は「check」ボタンをクリックすることによって、互いに素な概念に関するモデルの一貫性をチェックすることができる。

知識ベースの定義：実体、概念、関係の具体化 定義 2.3 において、実体 **I**、(関数 inst で得られる) 概念の具体化、(関数 instr で得られる) 関係の具体化から知識ベースの構造 **KB:=(O,I,inst,instr))** が導入された。

図 7.8 に、モデル化されたオントロジに加えて、知識ベースを定義するために **ONTOEDIT** によって支援されるビューを示す。

図 7.8 の左側には、概念の階層ビューが表示され、右側には概念の実体が列挙される。

オントロジの設計者は、実体ビューを用いて、概念階層から実体化すべき概念を選択する。図 7.8 の例では、概念 **PERSON** が実体化のために選択されている。概念階層においてある概念を選択することによって、図 7.8 における右上のウィンドウが起動する。概念 **PERSON** とそのサブ概念 (例えば **AUFSICHETSRATCHER**) のすべての定義された実体が列挙される。利用者は、例えば列挙された実体と他の定義済み実体を関連付けることによって、実体を変更することができる。

図 7.8 の右下に実体間の関係を変更、追加するためのダイアログを示す。このダイアログでは、オントロジに含まれる情報を利用する。例えば、概念 **PERSON** の実体はオントロジにおける概念 **PERSON** のために定義された関係を実体化するのみである。この小さな例では、関係 **HAT_NAME** が選択され、実体化されている。

もう1つの例は、関係 **GEHOERT_ORGANISATION_AN** を用いた関係の実体化である。この関係の実体化は、2つの実体である **ama:person** と **University_of_Karlstuhe** を結びつける。

F-Logic に基づく推定エンジンへのアクセス **ONTOEDIT** は2つの推論エンジンにアクセスすることができる。1つ目は F-LOGIC に基づく推論エンジン SiLRI である。2つ目は記述論理エンジン FACT である。ここでは、定義されたオントロジと知識ベースに対する F-LOGIC ベー

スのアクセスに焦点を当てる。全体のフレームワークの中で F-Logic エンジンがどのように利用されるかについて簡単に紹介する。

前に導入したように、オントロジ O と知識構造 KB から、F-Logic へのマッピングはすでに定義されている。加えて、意味的なパターンの中では捉えられていない F-Logic における規則や公理の定義が支援される。図 7.9 に、F-Logic 規則を定義し、アクセスするための F-Logic 規則エディタを示す。左側には、ON と OFF を切り替えることができる実体化された意味的パターンと自由に定義されたルールの F-Logic 変換を示す。

実体化されたオントロジと知識ベースの構造はオントロジの設計者によって検索されることもある。図 7.10 に、**ONTOEDIT** によって直にアクセス可能な **SiLRI F-Logic** 推論エンジンの検索ビューを示す。検索ビューを開くことによって、対応するオントロジ、また可能であれば知識ベースと共に推論エンジンの 1 つの実体を生成する。図 7.10 の右側の例では、次のような推論エンジンによって次のような F-Logic クエリが評価される。

(8)

このクエリによって、概念 **STUDENT** のすべての実体がその名前と電話番号によって検索される。その結果は、変数 z, k, e に代入される。

図 7.10 の左側は評価結果の説明を示す。この図は、例えばサブ概念の関係の推移性など、適切な代入を生成するために発火した規則を示す。(規則 2.0 を参照)

7.3 オントロジ学習のためのコンポーネント

前節では、手動によるオントロジの設計、つまりオントロジ設計環境 **ONTOEDIT** の包括的な操作と、グラフィカル・ユーザ・インタフェースの一部を説明した。本節では、**ONTOEDIT** をオントロジ学習環境 **TEXT-TO-ONTO** に拡張したコンポネントに焦点を当てる。

データのインポートと処理環境 前に導入したように、オントロジの設計者はデータのインポートとその処理から始める。図 7.11 にドキュメントの選択と 処理環境を示す。図 7.11 の左上に示されたボタンをクリックすることによって、利用者はドキュメントの番号を、[+Web] ボタンを押せば web から、[+File] ボタンを押せばローカルのファイル環境から、インデックス化することができる。web ボタンを選択すれば、オントロジに基づく、のろのろアルゴリズムが実行され、学習コーパスが Web から自動的に集められる。

図 7.11 の上の部分に示されたダイアログは、インデックス化されたドキュメント上の異なる操作の機能を許す。まず必要であれば、HTML の要素が削除され、書換え規則が適用される。次に、ドキュメントを正規化するための自然言語処理コンポーネントが、このダイアログからアクセスされる。

重要な点は、自然言語処理コンポーネントが実オントロジとその利用可能な背景知識のためのドキュメントを参照するための用語にアクセスするということである。

言語的な前処理をされたドキュメントは、変換モジュールへの入力として仕える。

ドキュメント概念の関係を生成するための表示の例はすでに図 5.12 に示した。他の利用可能なグラフィカル・コンポーネントは、半構造化ドキュメント、特にドメイン固有の辞書、をインポートするためのドキュメント・ラッパーである。

アルゴリズムのライブラリ アルゴリズムのライブラリは、適切に前処理されたデータを入力として受けとる。どのようにしてアルゴリズムが実行されるか、また、アルゴリズムアプリケーションに対する関連性のある情報がどのように定義されるかを次に示す。

パターン設計 テキストから階層構造、また非階層構造の関係を抽出するための正規表現に基づく機構が、1.2.2 節で導入された。**パターンデバッガ**を用いて、正規表現が反復的に発達され、例となるテキストを適用することによって、洗練される。存在するパターンは、パターンデバッガを用いて特定の型のテキストに適用される。

図 7.12 に、オントロジ学習に対する正規表現パターンの発展とデバッグを許可するパターン設計の図を示す。パターンの設計時に利用者を支援することは、その具体的な応用のために重要な点である。図 7.12 の左側に、環境のレポジトリに蓄積された、あるパターンが選択され、利用者に示されている。パターンは、パターンのカテゴリに属する、名前、短い説明を持ち、自然と具体的な正規表現を含む。

図 7.12 の右側に、パターン設計とデバッグのための、ビューを示す。ある特定の、選択されたパターンがテストドキュメントに適用され、対応する要素が利用者に提示される。このビューを用いて、利用者は個々のパターンを利用可能なドメインのテキストや個々のディレクトリに適用することができる。

用語エントリの抽出 用語エントリの抽出は、用語法の拡張と、新しい概念の提案のために基本となる機構である。

現在の実装では、利用者は 2 つの選択基準から選択することができる。そして用語エントリのために計算された基準が越えなければならない閾値を定義する。

階層的クラスタリング 階層的クラスタリングは、派生概念の階層、及び既存概念階層の拡張のための機構として導入された。階層的クラスタリングの問題は、異なるパラメータ群の下で使われる可能性があるということである。グラフィカルな手段が、異なる可能なパラメータを選択するため (例えば、類似関数と、類似基準) と、図 6.1 で説明したような命名規則によって利用されるかもしれない既存の背景知識を選択するために提供される。

非分類学的関係の抽出 6 章において、非分類学的関係の抽出のための異なる機構が導入された。この技術は用語法と概念レベルにおいて適用可能である。もし、概念 HC の分類学が利用可能であれば、学習はこの分類を背景知識として利用する。図 7.13 に、非分類学的関係を抽出するための機構をスタートさせるための、グラフィカルなインタフェースを示す。

図 7.13 の左側のダイアログは、利用者によって定義される可能性のある、異なる構成を与える。まず、アルゴリズムは、6 章の 1.3.3 節で述べられたように、特定の概念に対する制約であるかもしれない。

次に、利用者は、異なる支援と信頼度を定義する可能性がある。8 章において、抽出結果に対するこの値の変更が何を意味するのかが、説明される。

典型的に、利用者は一般的な関係を探索するために、高い支援と信頼度で始め、その後、より詳しい関係を探索するために、値を減らすという、経験が示された。

オントロジの刈り込み オントロジの刈り込みは、「逆」用語エントリの抽出として導入された。基本的刈り込みと、相対的刈り込みアルゴリズムは区別される。初めのケースでは、利用者はドキュメントや頻度や閾値の集合を選択する。このアルゴリズムはオントロジから削除される概念を提案する。二つ目のケースでは、利用者は、ドメイン固有または一般的なコーパスや頻度、閾値を選択する。再び、このテクニックの適用の結果として、刈り込まれるであろう、概念の集合が提案される。

オントロジの改良 オントロジの改良においては、その概念的なふるまいと、未知の用語エントリの既存概念へのマッピングの定義に沿って、重要な用語エントリの抽出に焦点が置かれる。利用者は関連するコーパス、類似基準、閾値 (k) を選択することによって、改良アルゴリズムを始める。

そして、繰返し、k 類似概念を持つ未知の用語エントリが検索され、利用者に提示される。

結果の表示 異なるアルゴリズムが同じまたは類似した結果を生成することが紹介されている。パラメータに依存して、このアルゴリズムは異なる結果を生成する。従って、抽出された結果を表示するための適切な機構が要求される。結果を表示するための異なる機構が、オントロジ学習フレームワークに実装されている。注目点は、結果表示コンポーネント (図 7.14) とグラフベースの視覚化 (図 7.15) に制約されている。

結果表示コンポーネント 結果表示とマッチングコンポーネントは、二つの異なるビューに分けられる。最初のビューは、1 つの実体を提示する。次のビューは、抽出された実体間の 2 項関係を示す (例えば、概念間の分類学的または非分類学的関係)。

図 7.14 に、これらの 2 つのビューを示す。図 7.14 の左に、概念のための検出された用語エントリを持つ 1 つの実体ビューを示す。図 7.14 の右に、実体間の 2 項関係を表すビューを示す。

グラフに基づく視覚化 グラフに基づく視覚化機構は、バネ埋め込み型アルゴリズムに基づく。図 7.15 に、非分類学的関係を抽出するためのアルゴリズムを適用することによって生成された、概念ネットワークの、ある視覚化を示す。

グラフに基づく視覚化は、利用者が二項関係を探索するだけでなく、概念間の関係を「繋ぐ」という利点を持つ。

7.4 まとめ

この章では、実装環境 **TEXT-TO-ONTO** の一部、すなわち、研究結果としてのオントロジ学習環境が示された。本章の初めからの引用に従って、既存の Web data からの「実世界の知識取得」**TEXT-TO-ONTO** が確かにうまく利用されました。

しかしながら、例によって、開発中のソフトウェア環境においては、オントロジの設計と学習環境の改良のために要求される多くの課題が存在する。次に、環境を拡張するための重要な可能性を描いている。

まず、前にも述べたように、**TEXT-TO-ONTO** 環境は、オントロジ設計をサポートするオントロジ学習を適用するときに、方法論的ガイドラインを提供するという点で、まだ初期の段階である。

従って、将来的に、半自動的方法への応用の支援を伴う大局的な方法論が要求される。これは特にデータのインポートと処理などの難しいタスクに対して維持する。

次に、現在のところ、オントロジ学習環境 **TEXT-TO-ONTO** はドイツ語に基づくテキストデータに制限される。

一般に、自然言語処理コンポーネントが他の NLP コンポーネント、例えば、英語等で置き換え、または補足されることを指摘した。

しかし、自然言語を処理する、異なるエンジンを利用することは、言語固有の問題から抽出するための、明確に定義された表現層を要求する。

多言語状況において **TEXT-TO-ONTO** を用いることは、異なる言語固有のエンジンによって生成される階層的 NLP の結果に対して、汎用的な技術の開発を必要とする。

さらに、柔軟なプラグイン機構が開発されるべきである。個々のアプリケーションは典型的に、オントロジ学習と設計環境の既存機能の拡張や適応を必要とする。プラグイン機構は特定の機能を提供するプラグインの位置と管理を支援するために開発された。

プラグインとサービス機構を持つ構造が、文献にある。

コンポーネントは動的にコア環境にプラグされる。プラグイン機構は、新しいコンポーネントが登録されたときに、インストールされているそれぞれのコンポーネントに通知する。サービス機構を通じて、そ

れぞれのコンポーネントは、他のコンポーネントによって与えられるサービスを発見し、利用できる。

コンポーネントによって表現されるサービスは典型的に典型的には、他のインタフェースへの参照である。

これは実装からのサービスの分離を提供し、代替実装を許可する。

最後に、セマンティック Web の開発と応用のための社会基盤カーネルとしてのサーバが要求される。オントロジの設計と学習環境は、いくつかの他のオントロジに基づくアプリケーションが要求する、多くの機能を含む。

厳密に言えば、本書で開発した設計環境は、セマンティック Web アプリケーションのための基盤カーネルとして動作するサーバにアクセスする、単なるクライアントである。このシステムは、コンポーネントベースであり、いくつかのコアなコンポーネントと密接に結びついた拡張可能なプラグイン構造として実現される。

他のサービスに対する基本としての RDF に基づく永続的ストレージに対するオントロジと実体レポジトリ、推論サービスを提供するための推論エンジン、バージョン管理、オントロジ設計に対する API、メンテナンス、移動、統合、そしてセマンティック Web アプリケーション。

このタスクに対する挑戦は、既に存在する、または、研究コミュニティによって提供されつつあるコンポーネントを互いに溶接することを許可する、有用かつオープンなインタフェースの提供である。

Chapter 8: EVALUATION

美濃研 D3 飯山将晃

2002 年 8 月 26 日 (月)

これまで述べてきたセマンティックウェブにおけるオントロジ学習の成果を評価することは非常に困難である．情報検索における精度や再現率，機械学習における正確さ (accuracy) などのように，スタンダードな評価尺度に相当するものがオントロジ学習に存在しないことがその理由である．

オントロジ学習を評価する方法として 2 つの大きなアプローチがあげられる．ひとつはアプリケーションに特化した評価方法であり，もうひとつが “gold-standard” を用いて 2 つのオントロジを相互比較することで，オントロジ学習の成果を評価する方法である．

本章では後者の評価方法について述べ，その評価方法に基づいて実験・評価を行う．

1 The Evaluation Approach（評価方法）

2 つのオントロジ $\mathcal{O}_1$,$\mathcal{O}_2$ を比較する方法としては以下にあげる 3 つのアプローチが挙げられる (図 8.1)．

1. 精度・再現率を用いて 2 つのオントロジがどれくらい一致しているかを計測する．
2. 2 つのオントロジの語彙レベルでの重複度や類似度を計測する．
3. 2 つのオントロジの概念レベルでの類似度を計測する．これは，2 つのオントロジで重複する語彙や，同一の語彙に対する概念レベルでの類似度を計算することによって計測することができる．

語彙レベルでの比較では，形式の類似度，言い換えると文字列の類似度しか計測することができないが，概念レベルでの比較ではより豊富な関係，例えば概念階層や非階層的関連性などを利用することが出来る．なお，オントロジ公理 $\mathcal{A}^{\mathcal{O}}$ は比較・評価の対象には含まない．

2 Ontology Comparison Measures（オントロジの比較基準）

2.1 Precision and Recall（精度と再現率）

精度 (precision) と再現率 (recall) は情報検索の分野においてよく用いられる評価尺度である．これらは，ある母集団から抽出された目的集合が，どの程度正確であるかを評価する際に用いられる．

一般的な設定としては，母集団が陽性ドキュメントと陰性ドキュメントで構成され，さらに真陽性（tp），真陰性（tn），偽陽性（fp），偽陰性（fn）に分割されている設定である．通常，抽出された目的集合は真陽性と偽陽性の要素で構成される（図 8.2）．

この設定に基づき，オントロジ学習における精度と再現率を次のように定義する

Definition 8.1 (Precision(精度)) 精度はシステムによって抽出された要素が正しいものである割合であり,

$$precision = \frac{tp}{tp + fp} \tag{2.1}$$

で定義される．オントロジ学習の場合，生成されたオントロジ（参照オントロジ Ref）の *"gold-standard"* オントロジ *(*比較オントロジ $Comp$*)* に対する精度は

$$precision_{OL} = \frac{|Comp \cap Ref|}{|Comp|} \tag{2.2}$$

のように定義される．

Definition 8.2 (Recall（再現率）) 再現率はシステムが抽出することができた要素の割合であり,

$$recall = \frac{tp}{tp + fn} \tag{2.3}$$

で定義される．オントロジ学習の場合,

$$recall_{OL} = \frac{|Comp \cap Ref|}{|Ref|} \tag{2.4}$$

のように定義される．

2.2 Lexical Comparison Level Measures（語彙レベルでの比較基準）

語彙レベルでの比較基準においては，上位レベルに存在する概念的な影響を無視して単に文字列の比較をすることで，2 つのオントロジの比較を行う．

文字列の比較は編集距離 (edit distance) に基づいて行われる．編集距離は，与えられた 2 つの文字列に対し，片方の文字列をもう片方の文字列へと編集する際に必要となる文字単位の操作（挿入，削除，変更）の最小数によって定義される．

例えば，"Top Hotel" と "Top_Hotel" の編集距離 (ed) は 1 であり，ed("Top Hotel","Top_Hotel") = 1 と表記する．(例が悪い...)

この編集距離に基づいて 2 つの語彙エントリ (lexical entry)(L_i, L_j) 間の類似度 (SM) を以下のように定義する．

Definition 8.3 (String Matching（SM）)

$$SM(L_i, L_j) := \max\left(0, \frac{\min(|L_i|, |L_j|) - ed(L_i, L_j)}{\min(|L_i|, |L_j|)}\right) \in [0, 1] \tag{2.5}$$

SM は 0 から 1 の値をとり，1 のとき完全に語彙エントリが一致することを表す．例えば SM("Top Hotel","Top_Hotel") $= \frac{7}{8}$ である．

SM は n 個の語彙エントリからなる集合 $\vec{l_i} := (l_i^1 \ldots l_i^n)$,$\vec{l_j} := (l_j^1 \ldots l_j^n)$ 同士の類似度として一般化することが出来る．

Definition 8.4 (String Matching for n-tuples)

$$SM(\vec{l_i}, \vec{l_j}) := \sqrt[n]{\Pi_{m:=1\ldots n} SM(l_i^m, l_j^m)} \in [0, 1] \tag{2.6}$$

また，2 つの語彙 $\mathcal{L}_1, \mathcal{L}_2$ の類似度は次のように定義される．

Definition 8.5 (Averaged String Matching)

$$\overline{SM}(\mathcal{L}_1, \mathcal{L}_2) := \frac{1}{|\mathcal{L}_1|} \sum_{L_i \in \mathcal{L}_1} \max_{L_j \in \mathcal{L}_2} SM(L_i, L_j) \quad (2.7)$$

$\overline{SM}(\mathcal{L}_1, \mathcal{L}_2)$ はあるオントロジ $\mathcal{O}_1$(ターゲット) が別のオントロジ $\mathcal{O}_2$(ソース) にどれくらいカバーされているかを表す尺度となる．また定義からも明らかのように，$\overline{SM}(\mathcal{L}_1, \mathcal{L}_2)$ と $\overline{SM}(\mathcal{L}_2, \mathcal{L}_1)$ とは異なる値となる．

SM はアンダースコアやハイフンの有無，単数形と複数形の違いなどの，語彙エントリの擬似的な相違の影響を緩和することが出来る．当然，“power” と “tower” のように概念レベルでまったく異なるものでも類似性があると判定されてしまうことがあるが，このようなノイズを含んでいてもそれなりに有用な評価尺度となる．

2.3 Conceptual Comparison Level Measures（概念レベルでの比較基準）

2 つのオントロジを概念レベルで比較する方法について述べる．オントロジの中で概念的なものを構築するものは，概念階層 $\mathcal{H}^{\mathcal{C}}{}_1, \mathcal{H}^{\mathcal{C}}{}_2$，非階層的な関連性 $\mathcal{R}_1, \mathcal{R}_2$ である．ここでは $\mathcal{H}^{\mathcal{C}}$,$\mathcal{R}$ に関係する語彙を利用して概念レベルでの比較を行う．

2.3.1 Comparing two taxonomies $\mathcal{H}^{\mathcal{C}}{}_1$, $\mathcal{H}^{\mathcal{C}}{}_2$

概念 C_1 の構造的な情報は conceptual cotopy(SC) で表される．これは，C_1 の上位概念と下位概念の集合であり次のように定義される．

Definition 8.6 (Conceptual Cotopy(SC))

$$SC(C_i, \mathcal{H}^{\mathcal{C}}) := \{C_j \in \mathcal{C} | \mathcal{H}^{\mathcal{C}}(C_i, C_j) \vee \mathcal{H}^{\mathcal{C}}(C_j, C_i) \vee C_i = C_j\} \quad (2.8)$$

また概念の集合に対する SC は以下のように定義される．

Definition 8.7 (Conceptual Cotopy for Sets of Concepts)

$$SC_C(\{C_1, \ldots, C_n\}, \mathcal{H}^{\mathcal{C}}) := \bigcup_{i:=1\ldots n} SC(C_i, \mathcal{H}^{\mathcal{C}}) \quad (2.9)$$

例として図 8.3 に示される概念階層の場合，$SC(\mathcal{F}(\{\text{“person”}\}), \mathcal{H}^{\mathcal{C}})$ は
$\mathcal{F}^{-1}(SC(\mathcal{F}(\{\text{“person”}\}), \mathcal{H}^{\mathcal{C}})) = \{\text{“student”}, \text{“researcher”}, \text{“person”}\}$ となる．

ある語彙エントリ L' からみたときの 2 つの概念階層 $\mathcal{H}^{\mathcal{C}}{}_1, \mathcal{H}^{\mathcal{C}}{}_2$ の重複 (TO) は SC を用いて以下のように定義される．

Definition 8.8 (Taxonomic Overlap(TO))

$$TO'(L', \mathcal{O}_1, \mathcal{O}_2) := \left\{ \frac{|\mathcal{F}_1^{-1}(SC(\mathcal{F}(\{L''\}), \mathcal{H}^{\mathcal{C}}{}_1)) \cap \mathcal{F}_2^{-1}(SC(\mathcal{F}(\{L''\}), \mathcal{H}^{\mathcal{C}}{}_2))|}{|\mathcal{F}_1^{-1}(SC(\mathcal{F}(\{L''\}), \mathcal{H}^{\mathcal{C}}{}_1)) \cup \mathcal{F}_2^{-1}(SC(\mathcal{F}(\{L''\}), \mathcal{H}^{\mathcal{C}}{}_2))|} \right\} \quad (2.10)$$

ここで定義した TO' をすべての語彙エントリに対して計算し，その平均を取ることで，2 つの概念階層間の類似度を計算することが出来る．但し，片方の語彙 $\mathcal{L}^{\mathcal{C}}{}_1$ の中に含まれるエントリ L'' が，もう片方の語彙 $\mathcal{L}^{\mathcal{C}}{}_2$ の中に含まれているとは限らない．対策として，L'' に対応する概念がもう片方の概念には存在しないと単純に仮定することもできるが，ここでは，$\mathcal{L}^{\mathcal{C}}{}_2$ に含まれるすべてのエントリに対し TO' を計算し，もっとも TO' の値が大きい (=重複している）エントリを $\mathcal{L}^{\mathcal{C}}{}_2$ における L'' とみなす．つまり，L'' からみたときの TO は次の最大化問題を解くことで与えられる．

Definition 8.9

$$TO''(L'', \mathcal{O}_1, \mathcal{O}_2) := \max_{C \in \mathcal{C}_2} \left\{ \frac{|\mathcal{F}_1^{-1}(SC(\mathcal{F}(\{L''\}), \mathcal{H}^{\mathcal{C}}{}_1)) \cap \mathcal{F}_2^{-1}(SC(C), \mathcal{H}^{\mathcal{C}}{}_2))|}{|\mathcal{F}_1^{-1}(SC(\mathcal{F}(\{L''\}), \mathcal{H}^{\mathcal{C}}{}_1)) \cup \mathcal{F}_2^{-1}(SC(C), \mathcal{H}^{\mathcal{C}}{}_2))|} \right\} \quad (2.11)$$

以上のことより，2 つのオントロジ $\mathcal{O}_1, \mathcal{O}_2$ に含まれる概念階層の類似度 $\overline{TO}$ は次の式で定義される．

Definition 8.10

$$\overline{TO}(\mathcal{O}_1, \mathcal{O}_2) := \frac{1}{|\mathcal{L}_1^C|} \sum_{L \in \mathcal{L}_1^C} TO(L, \mathcal{O}_1, \mathcal{O}_2) \quad (2.12)$$

$$TO(L, \mathcal{O}_1, \mathcal{O}_2) := \begin{cases} TO'(L, \mathcal{O}_1, \mathcal{O}_2) & if L \in \mathcal{L}_2^C \\ TO''(L, \mathcal{O}_1, \mathcal{O}_2) & if L \notin \mathcal{L}_2^C \end{cases} \quad (2.13)$$

例として，図 8.4 で示される 2 つのオントロジについて考えると，$TO'(\text{“hotel”}, \mathcal{H}^{\mathcal{C}}{}_1, \mathcal{H}^{\mathcal{C}}{}_2)$ は，$\mathcal{F}_1^{-1}(SC(\mathcal{F}(\{\text{“hotel”}\}), \mathcal{H}^{\mathcal{C}}{}_1)) = \{\text{“hotel”, “accomodation”}\}$, $\mathcal{F}_2^{-1}(SC(\mathcal{F}(\{\text{“hotel”}\}), \mathcal{H}^{\mathcal{C}}{}_2)) = \{\text{“wellness hotel”, “hotel”}\}$ より $\frac{1}{3}$ となる．また，$\mathcal{L}^{\mathcal{C}}{}_1$ にしか存在しない “accomodation” について考えると，$\mathcal{F}_1^{-1}(SC(\mathcal{F}(\{\text{“accomodation”}\}), \mathcal{H}^{\mathcal{C}}{}_1)) = \{\text{“youth hostel”, “accomodation”, “hotel”}\}$ であり，$\mathcal{L}^{\mathcal{C}}{}_2$ にある “hotel” が $\mathcal{F}_2^{-1}(SC(\mathcal{F}(\{\text{“hotel”}\}), \mathcal{H}^{\mathcal{C}}{}_2)) = \{\text{“wellness hotel”, “hotel”}\}$ となって，TO'' の最大値を与えるため $TO''(\text{“accomodation”}, \mathcal{H}^{\mathcal{C}}{}_1, \mathcal{H}^{\mathcal{C}}{}_2) = \frac{1}{4}$ となる．

$\overline{TO}$ には 2 つの特徴がある．

- $\overline{TO}$ は非対称である．つまり，$\overline{TO}(\mathcal{O}_1, \mathcal{O}_2)$ と $\overline{TO}(\mathcal{O}_2, \mathcal{O}_1)$ は一般的に異なる値をとる．
- $\mathcal{L}^{\mathcal{C}}{}_1$ と $\mathcal{L}^{\mathcal{C}}{}_2$ との関連性が全くない場合，$\overline{TO}$ は全く意味をなさなくなる．逆に言えば，$\mathcal{L}^{\mathcal{C}}{}_1$ と $\mathcal{L}^{\mathcal{C}}{}_2$ とがオーバーラップすればするほど，$\overline{TO}$ は 2 つの概念階層間の類似性を反映できる．

2.3.2 Comparing Non-Taxonomic Relations（非階層的関係の比較）

関係 R_1 は，語彙レベルでは語彙エントリ $L \in L^{\mathcal{R}}$ と対応付けられ，概念レベルでは 2 つの概念 C_1,C_2 と対応付けられる．従って，2 つの関係 R_1, R_2 の概念レベルでの関連性は 2 組のペア $R_1(C_1, C_2), R_2(C_3, C_4)$ を比較することで得られる．

2 つの関係の関連性は RO（relation overlap）によって得られる．これは，関係のドメインと範囲がどれくらい類似しているかを表す値である．

まず upwards cotopy(UC) を次のように定義する．

Definition 8.11 (Upwards Cotopy(UC))

$$UC(C_i, \mathcal{H}^{\mathcal{C}}) := \{C_j \in \mathcal{C} | \mathcal{H}^{\mathcal{C}}(C_i, C_j) \vee C_j = C_i\} \quad (2.14)$$

2 つの概念間の類似度 (concept match(CM)) はこの UC を用いて次のように表される．

Definition 8.12

$$CM(C_1, \mathcal{O}_1, C_2, \mathcal{O}_2) := \frac{|\mathcal{F}_1^{-1}(UC(C_1, \mathcal{H}^{\mathcal{C}}{}_1)) \cap \mathcal{F}_2^{-1}(UC(C_2, \mathcal{H}^{\mathcal{C}}{}_2))|}{|\mathcal{F}_1^{-1}(UC(C_1, \mathcal{H}^{\mathcal{C}}{}_1)) \cup \mathcal{F}_2^{-1}(UC(C_2, \mathcal{H}^{\mathcal{C}}{}_2))|} \quad (2.15)$$

例として，図 8.5 に示されるオントロジについて考えると，$UC(\mathcal{F}(\{\text{“researcher”}\}), \mathcal{H}^{\mathcal{C}})$ は $\mathcal{F}_1^{-1}(UC(\mathcal{F}(\{\text{“researcher”}\}), \mathcal{H}^{\mathcal{C}})) = \{\text{“researcher”}, \text{“person”}\}$ となり，$UC(\mathcal{F}(\{\text{“project”}\}), \mathcal{H}^{\mathcal{C}})$ は $\mathcal{F}_1^{-1}(UC(\mathcal{F}(\{\text{“project”}\}), \mathcal{H}^{\mathcal{C}})) = \{\text{“project”}\}$ となる．また，$\mathcal{F}(\{\text{“researcher”}\})$ と $\mathcal{F}(\{\text{“project”}\})$ との CM は 0 となり，$\mathcal{F}(\{\text{“researcher”}\})$ と $\mathcal{F}(\{\text{“student”}\})$ との CM は $\frac{1}{3}$ となる．

次に，UC と CM に基づいて非階層的関係の重複度 RO を定義する．

Definition 8.13

$$RO'(R_1, \mathcal{O}_1, R_2, \mathcal{O}_2) := \sqrt{d \cdot r} \tag{2.16}$$

ここで，$d := CM(domain(R_1), \mathcal{O}_1, domain(R_2), \mathcal{O}_2)$，$r := CM(range(R_1), \mathcal{O}_1, range(R_2), \mathcal{O}_2)$ である．

ある語彙エントリ L に対応する 2 つの関係間の重複度 RO'' は次の式で定義される．

Definition 8.14

$$RO''(L, \mathcal{O}_1, \mathcal{O}_2) := \frac{1}{|\mathcal{G}_1(\{L\})|} \sum_{R_1 \in \mathcal{G}_1(\{L\})} \{RO'(R_1, \mathcal{O}_1, R_2, \mathcal{O}_2)\} \tag{2.17}$$

また，L が $\mathcal{L}_2$ に含まれていない場合は，以下のようになる．

Definition 8.15

$$RO'''(L, \mathcal{O}_1, \mathcal{O}_2) := \frac{1}{|\mathcal{G}_1(\{L\})|} \sum_{R_1 \in \mathcal{G}_1(\{L\})} \max_{R_2 \in \mathcal{R}_2} \{RO'(R_1, \mathcal{O}_1, R_2, \mathcal{O}_2)\} \tag{2.18}$$

RO'' と RO''' を統合すると，ある語彙エントリ L からみたときの非階層的関係の類似度は次の式で定義される．

Definition 8.16

$$RO(L, \mathcal{O}_1, \mathcal{O}_2) := \begin{cases} RO''(L, \mathcal{O}_1, \mathcal{O}_2) & if L \in \mathcal{L}_2^{\mathcal{R}_2} \\ RO'''(L, \mathcal{O}_1, \mathcal{O}_2) & if L \notin \mathcal{L}_2^{\mathcal{R}_2} \end{cases} \tag{2.19}$$

以上のことより，2 つのオントロジ $\mathcal{O}_1, \mathcal{O}_2$ に含まれる非階層的関係の類似度 $\overline{RO}$ は次の式で定義される．

$$\overline{RO}(\mathcal{O}_1, \mathcal{O}_2) := \frac{1}{|\mathcal{L}^{\mathcal{R}_1}|} \sum_{L \in \mathcal{L}_1^R} RO(L, \mathcal{O}_1, \mathcal{O}_2) \tag{2.20}$$

例として図 8.6 で示される 2 つのオントロジについて考えた場合，“located at” で参照される関係はオントロジ $\mathcal{O}_1$ では “Hotel” と “AREA” との関係であり，オントロジ $\mathcal{O}_2$ では “Hotel” と “CITY” との関係である．“Hotel” からみたときの CM は $\frac{1}{2}$ であり，$\mathcal{O}_1$ での “area”,cO_2 での “city” が参照する概念の CM は $\frac{1}{2}$ となる．この結果，“located at” からみたときの RO' は $\sqrt{\frac{1}{2} \cdot \frac{1}{2}} = 0.5$ となり，これが $\overline{RO}$ の入力となる．

$\overline{RO}$ の特徴として，$\overline{RO}$ の値は 2 つのオントロジ間の語彙レベルでの類似性や概念階層レベルでの類似度によって左右されることが挙げられる．一般的には語彙と概念階層がある程度一致していないと，$\overline{RO}$ の値は大きくならない．

3 Human Performance Evaluation

本節では，人間が構築したオントロジに対する評価を行なう．主な評価事項は次の 2 つである．

- 実データに対してどの程度の評価が得られるか
- 異なる被験者によって構築されたオントロジがどの程度類似しているか

被験者には旅行に関する WWW サイトを用いてオントロジを構築してもらい，あらかじめ用意した評価用オントロジ $\mathcal{O}_{gold}$ と比較する．$\mathcal{O}_{gold}$ は 1200 の語彙エントリ，311 の概念，322 の概念階層，71 の非階層的関係で構成されている．

3.1 Ontology Engineering Evaluation Study

オントロジ学習を評価するために，実験を 3 つのフェーズに分け，評価を行った．

- Phase I 小規模の概念構造を被験者に与え，被験者には旅行のドメインに対するオントロジを構築してもらった．作成してもらうオントロジに対し，概念に対応する語彙エントリを 300 までに，関係に対応する語彙エントリを 80 までに制限した．
- Phase II $\overline{TO}$ や $\overline{RO}$ の結果を評価するため，語彙の差異の影響を取り除いて実験を行った．被験者は与えられた 310 の語彙エントリを用い，まず，概念階層を構築してもらい，つぎに 80 程度の語彙エントリで関係を構築してもらった．
- Phase III $\overline{RO}$ の結果を評価するため，概念階層と語彙の差異の影響を取り除いて実験を行った．310 の語彙エントリとそれらに対応する 310 の概念と概念階層を被験者に与え，80 程度の語彙エントリで関係を構築してもらった．

3.2 Human Evaluation – Precision and Recall

先に述べた 3 つのフェーズに対して精度と再現率を計算した．フェーズ I の実験で生成されたオントロジに対し，オントロジを構成する語彙エントリについての精度と再現率を計算した結果を図 8.8 に示す．

また，フェーズ II の実験で生成されたオントロジに対し，概念階層に含まれる要素に対する精度と再現率を計算した結果を図 8.9 に，フェーズ III の実験で生成されたオントロジに対し，非階層的関係に含まれる要素に対する精度と再現率を計算した結果を図 8.10 に示す．

フェーズ I の結果では精度と再現率の平均値は共に 0.2 程度と低い値になっており，人間は共通した語彙エントリを使う傾向にはないと考えることが出来る．このことは，同一の語彙エントリを用いたフェーズ II の実験結果が良好であることからも裏付けられる．また，フェーズ III の実験結果より非階層的関係については低い相関性しかないことがわかる．

3.3 Human Evaluation – Conceptual Comparison Level

フェーズ I の実験結果に対して $\overline{SM}$ を計算した結果を表 8.3 に示す．

この結果から，人間は関係についての語彙エントリより，概念についての語彙エントリで共通した語彙エントリを用いると考えることができる．なお，別の実験で，SM の値が 0.75 以上の場合その 2 つの単語は一般的に近い意味をもつということがわかっている．

3.4 Human Evaluation – Lexical Comparison Level

フェーズ I,II,III に対して $\overline{TO}$ と $\overline{RO}$ を計算した結果を表 8.5,8.6,8.7 に示す.

フェーズ I とフェーズ II における $\overline{TO}$ の相関は 0.58 となり，被験者同士が共通した概念階層を用いるかどうかは，あらかじめ語彙エントリが与えられているかどうかとは関係がないことがわかる.

また，フェーズ I と II, フェーズ I と III における $\overline{RO}$ の相関はそれぞれ 0.34，0.27，またフェーズ II と III における $\overline{RO}$ の相関は 0.16 となり，被験者はあらかじめ用意された語彙をもちいることは容易であるが，用意された概念階層をもちいることは非常に難しいということがわかる.

4 Ontology Learing Performance Evaluation

本章ではシステムのよって生成されたオントロジに対する評価を行う.

入力データやシステムのパラメータなどを変化させることによっていくつかのオントロジをシステムを用いて生成し，あらかじめ人間の手によって作成された評価用のオントロジ $\mathcal{O}_{gold}$ との比較を行う.

4.1 The Evaluation Setting

実験に用いるデータや評価用のオントロジ $\mathcal{O}_{gold}$ は，前節の実験で用いたものと同一のものを用いている.

4.2 Evaluation of Lexical Entry Extraction

10775 の語彙エントリを含むコーパスを実験に用いた．lef(6 章で説明されているはず) の平均値は 17，最大値は 9438 であり，Tfdf の平均値は 44，最大値は 3427 である.

図 8.12 に語彙エントリに対する精度と再現率を計算した結果を示す.

lef や tfidf に関係なく再現率は低い値となっている．このことから，人間の手によって作成されたオントロジー（=評価用オントロジ）はコーパスの語彙をうまく反映させたものではないと結論付けることが出来る.

4.3 Evaluation of Concept Hierarchy Extraction

背景知識の量 (small = 55 個の概念, medium = 110 個の概念, large = 211 個の概念)，計算方式 (average, complete, single）を変えて生成された 9 つのオントロジに対し概念階層に対する精度と再現率および $\overline{TO}$ を計算した．その結果を図 8.13，8.14 に示す.

4.4 Evaluation of Non-Taxonomic Relation Extraction

support と confidence の値（6 章で説明されているはず）を変えて生成されたオントロジに対し，生成された非階層的関係の数，$\overline{RO}$，再現率，精度を計算した結果を表 8.9 図 8.15 に示す.

5 Conclusion

本章では，オントロジ学習の評価方法について述べた．

人間が生成したオントロジを評価した結果，人間同士で共通したオントロジを作成する傾向にはないことが示された．このことからオントロジを作成するには，人間同士での協調作業が必要であることを示している．

また，システムが生成したオントロジを評価することで，与えられたコーパスやタスクに応じてオントロジエンジニアが最適な方法を選択することが可能となる．

Fast Computation of Concept Lattices Using Data Mining Techniques

Gerd Stumme,[1] Rafik Taouil,[2] Yves Bastide,[2]
Nicolas Pasquier,[2] Lotfi Lakhal[2]

[1] Technische Universität Darmstadt, Fachbereich Mathematik,
Schloßgartenstr. 7, D–64289 Darmstadt, Germany;
stumme@mathematik.tu-darmstadt.de
[2] Laboratoire d'Informatique (LIMOS), Université Blaise Pascal,
Complexe Scientifique des Cézeaux, 24 Av. des Landais,
F–63177 Aubière Cedex, France;
{taouil,bastide,pasquier,lakhal}@libd2.univ-bpclermont.fr

Abstract

We present a new algorithm called TITANIC for computing concept lattices. It is based on data mining techniques for computing frequent itemsets. The algorithm is experimentally evaluated and compared with B. Ganter's Next-Closure algorithm.

1 Introduction

Concept Lattices are used to represent conceptual hierarchies which are inherent in data. They are the core of the mathematical theory of Formal Concept Analysis (FCA). Introduced in the early 80ies as a formalization of the concept of 'concept' [18], FCA has over the years grown to a powerful theory for data analysis, information retrieval, and knowledge discovery [14]. In Artificial Intelligence (AI), FCA is used as a knowledge representation mechanism. In database theory, FCA has been extensively used for class hierarchy design and management [12, 17, 19, 9]. Its usefulness for the analysis of data stored in relational databases has been demonstrated with the commercially used management system TOSCANA for Conceptual Information Systems [16].

A current research domain common to both the AI and the database community is Knowledge Discovery in Databases (KDD). Here FCA has been used as a formal framework for implication and association rules discovery and reduction [4, 11, 15] and for improving the response times of algorithms for mining association rules [10, 11]. The interaction of FCA and KDD in general has been discussed in [13] and [5].

In this paper we show that, vice versa, FCA can also benefit from ideas used for mining association rules: Computing concept lattices is an important issue, investigated for long years [9, 2, 4, 19]. We address the problem of computing concept lattices from a data mining viewpoint by using a level-wise approach [1, 8]; and provide a new, efficient algorithm called TITANIC.

In the next section, we present the theoretical foundation. It is turned into pseudo-code in Section 3. We conclude the paper with results of an experimental evaluation. Because of lack of space we will not provide proofs, and will not discuss the general use of FCA in AI and database theory.

2 Computing Concept Lattices by Using the Support Function

2.1 Formal Concept Analysis

In the first part of this section, we briefly recall the basic notions of Formal Concept Analysis. For a more extensive introduction into Formal Concept Analysis refer to [3].

Definition 1 *A* formal context *is a triple* $\mathbb{K} := (G, M, I)$ *where* G *and* M *are sets and* $I \subseteq G \times M$ *is a binary relation. The elements of* G *are called* objects *and the elements of* M items. *The inclusion* $(g, m) \in I$ *is read as "object* g *has attribute* m*". In this paper we assume that all sets are finite, especially* G *and* M*.*

For $A \subseteq G$*, we define* $A' := \{m \in M \mid \forall g \in A\colon (g, m) \in I\}$*; and for* $B \subseteq M$*, we define dually* $B' := \{g \in G \mid \forall m \in B\colon (g, m) \in I\}$*.* *A* formal concept *is a pair* (A, B) *with* $A \subseteq G$*,* $B \subseteq M$*,* $A' = B$ *and* $B' = A$*.* A *is called* extent *and* B *is called* intent *of the concept. The set of all concepts of a formal context* $\mathbb{K}$ *together with the partial order* $(A_1, B_1) \leq (A_2, B_2) :\Leftrightarrow A_1 \subseteq A_2$ *(which is equivalent to* $B_1 \supseteq B_2$*) is a complete lattice, called* concept lattice *of* $\mathbb{K}$*.*

	Jacobs	Plus	classic	mild	light	< 6 DM	< 8 DM	> 8 DM
Dallmayr Prodomo			×					×
Jacobs Krönung	×		×				×	
Jacobs Krönung Light	×				×		×	
Jacobs Krönung Free	×				×		×	
Jacobs Krönung Mild	×			×			×	
Jacobs Meisterröstung	×		×				×	
Tempelmann			×					×
Plus Schonkaffee		×			×		×	
Plus Naturmild		×		×		×	×	
Plus milde Sorte		×		×		×	×	
Plus Gold		×	×			×	×	
Idee Kaffee Classic			×					×
Kaffee Hag klassisch			×					×
Melitta Cafe Auslese			×				×	
Melitta Cafe Auslese Mild				×			×	
Kaisers Kaffee Auslese Mild				×			×	

Figure 1: Formal context about coffee brands sold in a supermarket.

Figure 1 shows a formal context which lists all coffee brands sold in a supermarket. Figure 2 shows the concept lattice of the context by a line diagram. In the *line diagram*, the name of an object g is always attached to the circle representing the

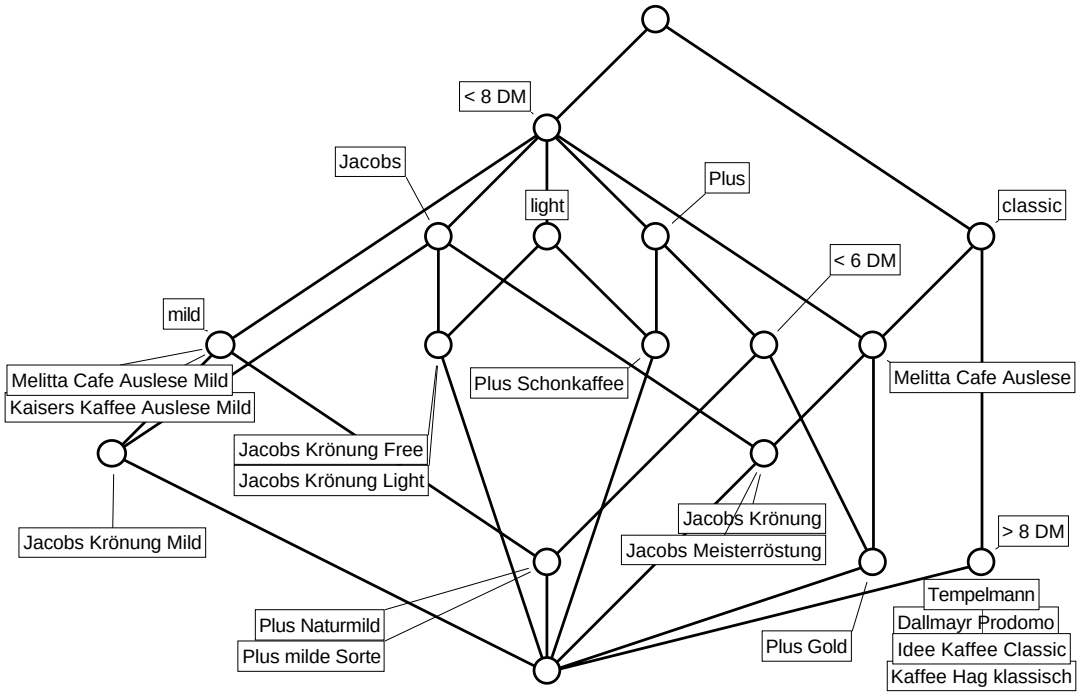

Figure 2: The concept lattice of the context in Figure 1

smallest concept with g in its extent; dually, the name of an attribute m is always attached to the circle representing the largest concept with m in its intent. This allows us to read the context relation from the diagram because an object g has an attribute m if and only if there is an ascending path from the circle labeled by g to the circle labeled by m. The extent of a concept consists of all objects whose labels are below in the diagram, and the intent consists of all attributes attached to concepts above in the hierarchy. For example, the concept labeled by '< 6 DM' has {'Plus Naturmild', 'Plus milde Sorte', 'Plus Gold'} as extent, and {'< 6 DM', 'Plus' [the house brand of the supermarket], '< 8 DM'} as intent.

For $X, Y \subseteq M$, we say that the *implication* $X \to Y$ *holds* in the context, if each object having all attributes in X also has all attributes in Y (i. e., an implication is an association rule with 100% confidence). For instance, the implication {Plus, classic} $\to$ {< 6 DM} holds in the coffee context. Implications can be read directly in the line diagram: the largest concept having both 'Plus' and 'classic' in its intent is below the concept labeled by '< 6 DM'. In [15] is shown how also the association rules with less than 100 % confidence can by visualized in the line diagram.

2.2 Support-Based Computation of the Closure System of all Concept Intents

In the following, we will use the composed function $\cdot'' : \mathfrak{P}(M) \to \mathfrak{P}(M)$ which is a closure operator on M (i. e., it is extensive, monotonous, and idempotent). The

related closure system (i. e., the set of all $B \subseteq M$ with $B'' = B$) is exactly the set of the intents of all concepts of the context. The structure of the concept lattice is already determined by this closure system. Hence we restrict ourselves to the computation of the closure system of all concept intents in the sequel. The computation makes extensive use of the following *support* function:

Definition 2 *The* support *of* $X \subseteq M$ *is defined by* $\mathrm{supp}(X) := \frac{|X'|}{|G|}$.

In the case of X and Y with $X'' = Y''$, both sets have obviously the same support. On the other hand, comparable attribute sets with the same support also have the same closures:

Lemma 1 *Let* $X, Y \subseteq M$.
(i) $X'' = Y'' \Longrightarrow \mathrm{supp}(X) = \mathrm{supp}(Y)$
(ii) $X \subseteq Y \wedge \mathrm{supp}(X) = \mathrm{supp}(Y) \Longrightarrow X'' = Y''$

This lemma allows us to develop the algorithm in a more general setting:

Definition 3 *A* weight function *on* $\mathfrak{P}(M)$ *is a function* $s\colon \mathfrak{P}(M) \to P$ *from the powerset of* M *to a partially ordered set* $(P, \leq)$. *For a set* $X \subseteq M$, $s(X)$ *is called the weight of* X. *The weight function is* compatible with a closure operator h *if* (i) $X \subseteq Y \Longrightarrow s(X) \geq s(Y)$,[1] (ii) $h(X) = h(Y) \Longrightarrow s(X) = s(Y)$, (iii) $X \subseteq Y \wedge s(X) = s(Y) \Longrightarrow h(X) = h(Y)$.

Let h be a closure operator on a finite set M, and let s be a compatible weight function. The task is now to determine efficiently the closure system $\mathcal{H} := \{X \subseteq M \mid h(X) = X\}$ related to the closure operator h.

It is easy to check, that for a given formal context, the support function fulfills the conditions of Definition 3 for the closure operator $h(X) := X''$. Another problem where such a weight function can be used is the computation of the closure system induced by those functional dependencies which are valid for the actual data of a relational database (refer to [6]).

We discuss the problem of computing the closure system by using a weight function in three parts:

1. How can we compute the closure of a given set using the weight function only, and not the closure operator?

2. How can we compute the closure system by computing as few closures as possible?

3. Since the weight function is usually not stored explicitly, how can we derive the weights of as many sets as possible from the weights already computed?

Questions 2 and 3 are not independent from each other. Hence we will not provide an optimal answer for each of them, but one which improves the overall benefit.

[1] If $X \subseteq Y \Longrightarrow s(X) \leq s(Y)$ holds instead of (i) (as e. g. for functional dependencies), then all 'min' in the sequel (beside the one in Definition 6) have to be replaced by 'max'.

2.2.1 Weight-based computation of closures

We use the constraints on the function s for determining the closure of a set by comparing its weight with the weights of its immediate supersets.

Proposition 2 *Let $X \subseteq M$. Then*

$$h(X) = X \cup \{m \in M \setminus X \mid s(X) = s(X \cup \{m\})\} \ .$$

Hence if we know the weights of all sets, then we can compute the closure operator ($\rightarrow$ Algorithm 3, steps 3–7).[2] In the next subsection we discuss for which sets it is necessary to compute the closure in order to obtain all closed sets. In Subsection 2.2.3 we discuss how the weights needed for those computations can be determined.

2.2.2 A level-wise approach for computing all closed sets

One can now compute the closure system $\mathcal{H}$ by applying Proposition 2 to all subsets X of M. But this is not efficient, since many closed sets will be determined several times.

Definition 4 *We define an equivalence relation θ on the powerset $\mathfrak{P}(M)$ of M by $(X, Y) \in \theta :\Longleftrightarrow h(X) = h(Y)$, for $X, Y \subseteq M$. The equivalence class of X is given by $[X] := \{Y \subseteq M \mid (X, Y) \in \theta\}$.*

If we knew the equivalence relation θ in advance, it would be sufficient to compute the closure for one set of each equivalence class only. But since we have to determine the relation during the computation, we have to consider more than one element of each class in general. As known from algorithms for mining association rules, we will use a level-wise approach.

Definition 5 *A k-*set *is a subset X of M with $|X| = k$. For $\mathcal{X} \subseteq \mathfrak{P}(M)$, we define $\mathcal{X}_k := \{X \in \mathcal{X} \mid |X| = k\}$.*

At the kth iteration, the weights of all k-sets which remained from the pruning strategy described below are determined; and the closures of all $(k-1)$-sets which passed the pruning in the $(k-1)$th iteration are computed.

The first sets of an equivalence class that we reach using such a level-wise approach are the minimal sets in the class:

Definition 6 *A set $X \subseteq M$ is a* key set *(or* minimal generator*) if $X \in \min[X]$. The set of all key sets is denoted by $\mathcal{K}$.*

Obviously we have $\mathcal{H} = \{h(X) \mid X \in \mathcal{K}\}$.

Proposition 3 *The set $\mathcal{K}$ is an order ideal of $(\mathfrak{P}(M), \subseteq)$ (i.e., $X \in \mathcal{K}$, $Y \subseteq X \Longrightarrow Y \in \mathcal{K}$).*

[2] In this section, we give some references to the algorithms in the following section. These references can be skipped at the first reading.

We will use a pruning strategy given in [1]. Originally this strategy was presented as a heuristic for determining all frequent sets only (which are, in our terminology, all sets with weights above a user-defined threshold). We show that this strategy can be applied to arbitrary order ideals of the powerset of M:

Definition 7 *Let $\mathcal{I}$ be an order ideal of $\mathfrak{P}(M)$. A* candidate set *for $\mathcal{I}$ is a subset of M such that all its proper subsets are in $\mathcal{I}$.*

This definition is justified by the following lemma:

Lemma 4 *Let $\mathcal{X} \subseteq \mathfrak{P}_k(M)$, and let $\mathcal{Y}$ be the set of all candidate $(k+1)$-sets for the order ideal $\downarrow \mathcal{X}$ (i. e., the order ideal generated by $\mathcal{X}$).*

1. *For each subset $\mathcal{Z}$ of $\mathcal{Y}$, there exists an order ideal $\mathcal{I}$ of $\mathfrak{P}(M)$ with $\mathcal{I}_k = \mathcal{X}$ and $\mathcal{I}_{k+1} = \mathcal{Z}$.*
2. *For each order ideal $\mathcal{I}$ of $\mathfrak{P}(M)$ with $\mathcal{I}_k = \mathcal{X}$ the inclusion $\mathcal{I}_{k+1} \subseteq \mathcal{Y}$ holds.*

The efficient generation of the set of all candidate sets for the next level is described in the following proposition ($\rightarrow$ Algorithm 2). We assume that M is linearly ordered, e. g., $M = \{1, \ldots, n\}$.

Proposition 5 *Let $\mathcal{X} \subseteq \mathfrak{P}_{k-1}(M)$. Let $\widetilde{\mathcal{C}} := \{\{x_1, \ldots, x_k\} \mid i < j \Longrightarrow x_i < x_j, \{x_1, \ldots, x_{k-2}, x_{k-1}\}, \{x_1, \ldots, x_{k-2}, x_k\} \in \mathcal{X}\}$; and $\mathcal{C} := \{X \in \widetilde{\mathcal{C}} \mid \forall x \in X : X \setminus \{x\} \in \mathcal{X}\}$. Then $\mathcal{C} = \{X \in \mathfrak{P}_k(M) \mid X$ is candidate set for $\downarrow \mathcal{X}\}$.*

Unlike in the Apriori algorithm, in our application the pruning of a set ($\rightarrow$ Algorithm 1, step 8) cannot be determined by its properties alone, but properties of its subsets have to be taken into account as well. This causes an additional step in the generation function ($\rightarrow$ Algorithm 2, step 5) compared to the original version presented in [1]. Based on this additional step, at each iteration the non-key sets among the candidate sets are pruned by using (ii) of the following proposition.

Proposition 6 *Let $X \subseteq M$.*
(i) *Let $m \in X$. Then $X \in [X \setminus \{m\}]$ if and only if $s(X) = s(X \setminus \{m\})$.*
(ii) *X is a key set if and only if $s(X) \neq \min_{m \in X}(s(X \setminus \{m\}))$.*

2.2.3 Deriving weights from already known weights

If we reach a k-set which is known not to be a key set, then we already passed along at least one of the key sets in its equivalence class in an earlier iteration. Hence we already know its weight. Using the following proposition, we determine this weight by using only weights already computed.

Proposition 7 *If X is not a key set, then*

$$s(X) = \min\{s(K) \mid K \in \mathcal{K}, K \subseteq X\} \ .$$

Hence it is sufficient to compute the weights of the candidate sets only (by calling a function depending on the specific implementation $\rightarrow$ Algorithm 1, step 7). All other weights can be derived from those weights.

3 The TITANIC Algorithm

The pseudo-code is given in Algorithm 1. A list of notations is provided in Table 1.

Algorithm 1 TITANIC

1) $\emptyset.s \leftarrow 1$;
2) $\mathcal{K}_0 \leftarrow \{\emptyset\}$;
3) $k \leftarrow 1$;
4) **forall** $m \in M$ **do** $\{m\}.p\_s \leftarrow 1$;
5) $\mathcal{C} \leftarrow \{\{m\} \mid m \in M\}$;
6) **loop begin**
7) WEIGH($\mathcal{C}$);
8) $\mathcal{K}_k \leftarrow \{X \in \mathcal{C} \mid X.s \neq X.p\_s\}$;
9) **forall** $X \in \mathcal{K}_k$ **do** X.closure $\leftarrow$ CLOSURE(X);
10) **if** $\mathcal{K}_k = \emptyset$ **then exit loop** ;
11) $k++$;
12) $\mathcal{C} \leftarrow$ TITANIC-GEN($\mathcal{K}_{k-1}$);
13) **end loop** ;
14) **return** $\bigcup_{i=0}^{k-1}\{X.\text{closure} \mid X \in \mathcal{K}_i\}$.

Table 1: Notations used in TITANIC

k	is the counter which indicates the current iteration. In the kth iteration, all key k-sets are determined.
$\mathcal{K}_k$	contains after the kth iteration all key k-sets K together with their weight $K.s$ and their closure K.closure.
$\mathcal{C}$	stores the candidate k-sets C together with a counter $C.p\_s$ which stores the minimum of the weights of all $(k-1)$-subsets of C. The counter is used in step 8 to prune all non-key sets.

The algorithm starts with stating that the empty set is always a key set, and that its weight is — in the case of concept lattices — always equal to 1 (steps 1+2). Then all 1-sets are candidate sets by definition (steps 4+5). In later iterations, the candidate k-sets are determined by the function TITANIC-GEN (step 12/Algorithm 2) which is (except step 5) a straight-forward implementation of Proposition 5. (The result of step 5 will be used in step 8 of Algorithm 1 for pruning the non-key sets.)

Once the candidate k-sets are determined, the function WEIGH($\mathcal{X}$) is called to compute, for each $X \in \mathcal{X}$, the weight of X and stores it in the variable $X.s$ (step 7). In the case of concept lattices, WEIGH determines the weights (i. e., the supports) of all $X \in \mathcal{X}$ *with a single pass* of the context. This is (together with the fact that only $\max\{|X| \mid X \subseteq M \text{ is candidate set}\}$ passes are needed) the reason for the efficiency of TITANIC.

Algorithm 2 TITANIC-GEN

Input: $\mathcal{K}_{k-1}$, the set of key $(k-1)$-sets K with their weight $K.s$.

Output: $\mathcal{C}$, the set of candidate k-sets C
with the values $C.p\_s := \min\{s(C \setminus \{m\} \mid m \in C\}$.

The variables p_s assigned to the sets $\{p_1, \ldots, p_k\}$ which are generated in step 1 are initialized by $\{p_1, \ldots, p_k\}.p\_s \leftarrow 1$.

1) $\mathcal{C} \leftarrow \{\{p_1, \ldots, p_k\} \mid i < j \Longrightarrow p_i < p_j,$
$\{p_1, \ldots, p_{k-2}, p_{k-1}\}, \{p_1, \ldots, p_{k-2}, p_k\} \in \mathcal{K}_{k-1}\}$;
2) **forall** $X \in \mathcal{C}$ **do begin**
3) **forall** $(k-1)$-subsets S of X **do begin**
4) **if** $S \notin \mathcal{K}_{k-1}$ **then begin** $\mathcal{C} \leftarrow \mathcal{C} \setminus \{X\}$; **exit forall** ; **end**;
5) $X.p\_s \leftarrow \min(X.p\_s, S.s)$;
6) **end**;
7) **end**;
8) **return** $\mathcal{C}$.

Algorithm 3 CLOSURE(X) for $X \in \mathcal{K}_{k-1}$

1) $Y \leftarrow X$;
2) **forall** $m \in X$ **do** $Y \leftarrow Y \cup (X \setminus \{m\})$.closure;
3) **forall** $m \in M \setminus Y$ **do begin**
4) **if** $X \cup \{m\} \in \mathcal{C}$ **then** $s \leftarrow (X \cup \{m\}).s$
5) **else** $s \leftarrow \min\{K.s \mid K \in \mathcal{K},\ K \subseteq X \cup \{m\}\}$;
6) **if** $s = X.s$ **then** $Y \leftarrow Y \cup \{m\}$
7) **end**;
8) **return** Y.

In step 8 of Algorithm 1, all candidate k-sets which are not key sets are pruned according to Proposition 6 (*ii*). For the remaining sets (which are now known to be key sets) their closures are computed (step 9). The CLOSURE function (Algorithm 3) is a straight-forward implementation of Propositions 3 and 7 (beside an additional optimization (step 2)). Algorithm 1 terminates, if there are no key k-sets left (step 11+14). Otherwise the next iteration begins (steps 10+12).

4 Experimental Evaluation and Conclusion

Several algorithms have been proposed for computing concept lattices. The most efficient at the best of our knowledge is Ganter's Next-Closure algorithm [2]. For our experimental evaluation, a version of the Titanic algorithm was implemented in C++ together with a rewriting of Ch. Lindig's C version of Next-Closure [7]. The comparisons took place on a Pentium III running at 600 MHz, with 512 MB of main memory, and were performed on the MUSHROOM (8,416 objects, 80 attributes) and

INTERNET (10,000 objects, 141 attributes) databases, both available from the *UCI KDD Archive* (`http://kdd.ics.uci.edu/`), with a varying number of objects.

The results are listed in Table 2 and visualized in Figure 3. They show that on the relatively strongly corelated MUSHROOMS database, Next-Closure is faster for few attributes, but takes twice the time of TITANIC for the whole dataset. On the weakly corelated INTERNET database, the difference is much larger. This stems from the fact that the development of TITANIC was inspired by the Apriori algorithm which is known to perform well on weakly corelated data.

Table 2: Database characteristics and evaluation results

Database	# of objects	# of attr.	# of concepts	Computation time (sec.) Next-Closure	Titanic
Mushrooms	2,500	79	5,394	31.13	48.19
	5,000	79	9,064	108.38	75.59
	8,416	80	32,086	527.74	200.73
Internet	1,000	141	15,107	16.49	4.33
	2,000	141	31,719	66.32	7.31
	5,000	141	73,026	381.95	14.31
	7,500	141	100,706	803.17	19.13
	10,000	141	124,574	1431.86	23.46

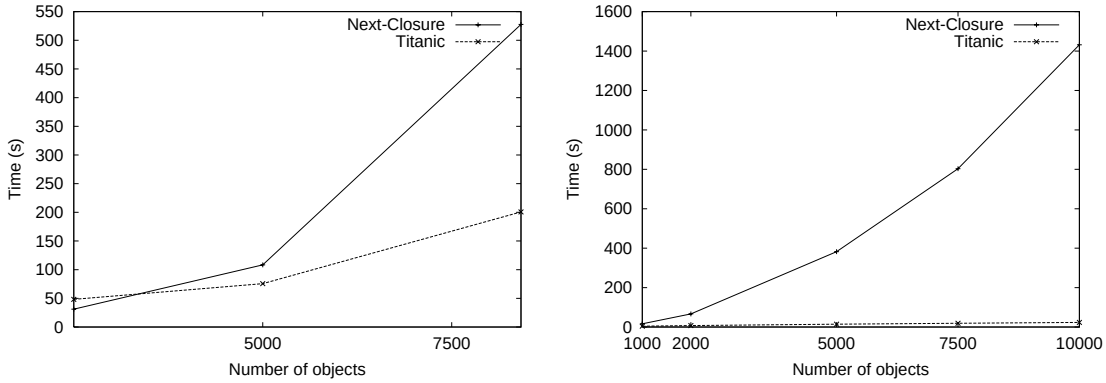

Figure 3: Comparison of TITANIC and Next-Closure on the MUSHROOMS (left) and INTERNET databases (right)

The problem of computing concept lattices has exponential complexity. This shows that one cannot expect from any algorithm — however robust it is claimed to be — that it solves the problem in reasonable time in the worst case. However our experimental results show that under normal conditions (and if handled with care) a strong and waterproof algorithm may improve the exploration of unknown regions of knowledge.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proc. VLDB Conf.*, 1994, 478–499 (Expanded version in IBM Report RJ9839)

[2] B. Ganter, K. Reuter: Finding all closed sets: A general approach. *Order*. Kluwer Academic Publishers, 1991, 283–290

[3] B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical Foundations*. Springer, Heidelberg 1999

[4] R. Godin, R. Missaoui: An incremental concept formation approach for learning from databases. *TCS* **133**(2): 387–419 (1994)

[5] J. Hereth, G. Stumme, U. Wille, R. Wille: Conceptual Knowledge Discovery and Data Analysis. In: B. Ganter, G. Mineau (eds.): *Conceptual Structures: Logical, Linguistic, and Computational Structures*. Proc. ICCS 2000. Springer, Heidelberg 2000 (to appear)

[6] J. Kivinen, H. Mannila: Approximate inference of functional dependencies from relations. *TCS* **149**(1): 129–149 (1995)

[7] Ch. Lindig: Concepts. ftp://ftp.ips.cs.tu-bs.de/pub/local/softech/misc/concepts-0.3d.tar.gz, 1997. (Open Source implementation of concept analysis in C)

[8] H. Mannila, H. Toivonen: Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining and Knowledge Discovery* **1**(3): 241-258 (1997)

[9] M. Missikoff, M. Scholl: An algorithm for insertion into a lattice: application to type classification. *Proc. 3rd Intl. Conf. FODO 1989*. LNCS **367**, Springer, Heidelberg 1989, 64–82

[10] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Discovering frequent closed itemsets for association rules. *Proc. ICDT Conf.*, 1999, 398–416

[11] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Efficient mining of association rules using closed itemset lattices. *Journal of Information Systems*, **24**(1), 1999, 25–46

[12] Ingo Schmitt, Gunter Saake: Merging inheritance hierarchies for database integration. *Proc. 3rd IFCIS Intl. Conf. on Cooperative Information Systems*, New York City, Nework, USA, August 20-22, 1998, 122–131

[13] G. Stumme, R. Wille, U. Wille: Conceptual Knowledge Discovery in Databases Using Formal Concept Analysis Methods. In: J. M. Żytkow, M. Quafofou (eds.): *Principles of Data Mining and Knowledge Discovery*. Proc. 2nd European Symposium on PKDD'98, LNAI **1510**, Springer, Heidelberg 1998, 450–458

[14] G. Stumme, R. Wille (eds.): *Begriffliche Wissensverarbeitung – Methoden und Anwendungen*. Springer, Heidelberg 2000

[15] R. Taouil, N. Pasquier, Y. Bastide, G. Stumme, L. Lakhal: Mining bases for association rules based on formal concept analysis. *Journal on Knowledge and Information Systems* (submitted)

[16] F. Vogt, R. Wille: TOSCANA – A graphical tool for analyzing and exploring data. LNCS **894**, Springer, Heidelberg 1995, 226–233

[17] K. Waiyamai, R. Taouil, L. Lakhal: Towards an object database approach for managing concept lattices. *Proc. 16th Intl. Conf. on Conceptual Modeling*, LNCS **1331**, Springer, Heidelberg 1997, 299-312

[18] R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.). *Ordered sets*. Reidel, Dordrecht–Boston 1982, 445–470

[19] A. Yahia, L. Lakhal, J. P. Bordat, R. Cicchetti: iO2: An algorithmic method for building inheritance graphs in object database design. *Proc. 15th Intl. Conf. on Conceptual Modeling*. LNCS **1157**, Springer, Heidelberg 1996, 422–437